Vol.7 No.1 May 1988

BEEBUG

FOR THE BBC MICRO & MASTER SERIES

Multi-Column Page Printer

- FILE HANDLING FOR ALL
- CAR PARK GAME
- DISC SPOOLER UTILITY
- EXTENDED VECTORS

BEEBUG Vol.7 No.1 May 1988

FEATURES

REVIEWS

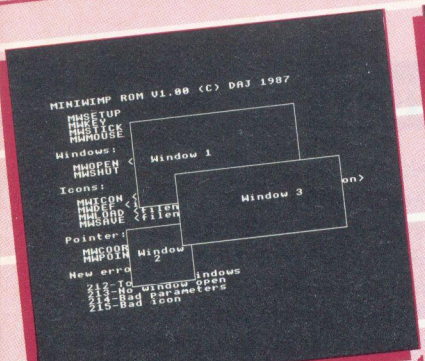
REGULAR ITEMS

HINTS & TIPS

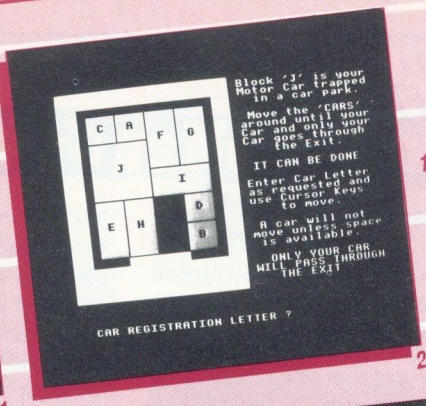
PROGRAM INFORMATION

All programs listed in BEEBUG magazine are produced direct from working programs. They are listed in LISTO1 format with a line length of 40. However, you do not need to enter the space after the line number when typing in programs, as this is only included to aid readability. The line length of 40 will help in checking programs listed on a 40 column screen.

Programs are checked against all standard Acorn systems (model B, B+, Master, Compact and Electron; Basic I and Basic II; ADFS, DFS and Cassette filing systems; and the Tube). We hope that the classification symbols for programs, and also reviews, will clarify matters with regard to compatibility. The complete set of icons is given



1.

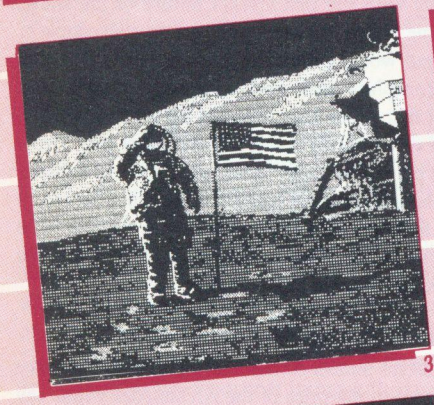


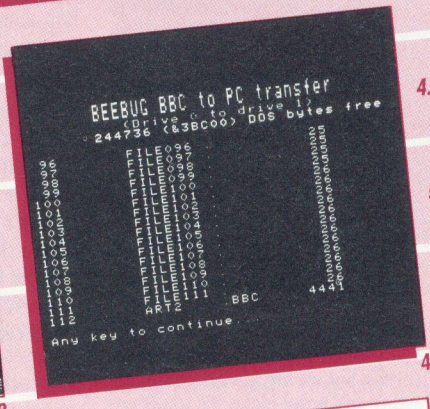
2.

1. BEEBUG Mini-Wimp

2. Boxed in the Carpark

3. Conquest Reviewed


3.

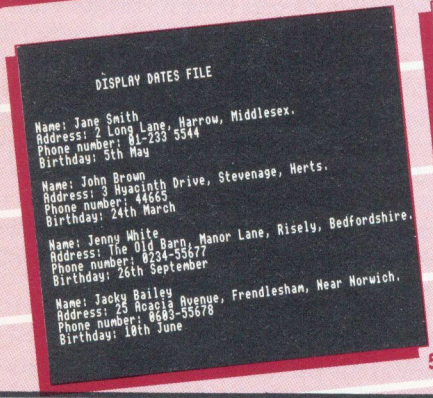


4.

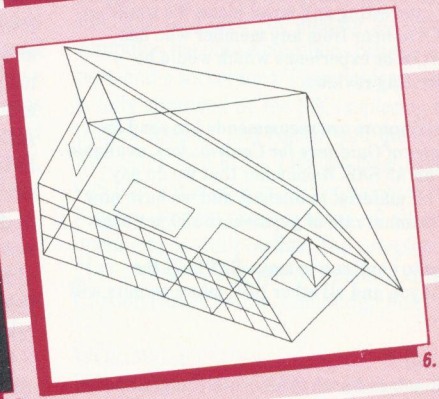
4. BBC to IBM File Transfer

5. File Handling

6. Super Dump



5.


6.

below. These show clearly the valid combinations of machine (version of Basic) and filing system for each item, and Tube compatibility. A single line through a symbol indicates partial working (normally just a few changes will be needed); a cross shows total incompatibility. Reviews do not distinguish between Basic I and II.

Computer System		Filing System	
Master (Basic IV)	M	ADFS	A
Compact (Basic VI)	C	DFS	D
Model B (Basic II)	II	Cassette	
Model B (Basic I)	I	**Tube Compatibility**	
Electron	E	Tube	

Editor's Jottings

CONTRIBUTING TO BEEBUG

Most of the articles and programs which we publish in BEEBUG are contributed by BEEBUG members. Indeed, many of the best and most interesting programs which we have published in the past have originated in this way. We also have a small band of more regular and experienced contributors, particularly where reviews are concerned.

We are currently seeking new material for publication in the magazine. Programs with or without accompanying explanation, short or long: all are welcome provided you believe that what you have to offer matches up to the standard of previously published material. If you have ideas for a series of several articles then we would urge you to contact us first to discuss this before undertaking too much detailed work.

At the present time we are particularly keen to receive applications and utilities, but we are always willing to consider any interesting and novel ideas. We would also be pleased to hear from any member who has specific expertise or experience which would be relevant for writing reviews.

Potential contributors are recommended to send for our leaflet *Notes of Guidance for Contributors*, available on receipt of an A5 SAE. Remember that we do pay promptly for all material published, and we have now raised our maximum rate of payment to £50 per page.

Please help us to continue to make BEEBUG the magazine that you and all other BBC micro owners will want to read.

VOLUME 6 INDEX

You will find included with this issue a complete index to volume 6 of BEEBUG. This has been organised to provide the maximum help when searching for a previous article. With six complete volumes, our computerised bibliography Magscan makes even more sense, and can greatly speed up finding all references to a particular subject. If you have not been keeping up-to-date with the monthly Magscan updates on the magazine discs, a complete volume 6 bibliography is being included on the magazine disc for this month at no extra charge. See inside back cover for details.

News News News Ne

SHOW CANCELLED

As speculated in the last BEEBUG, the Acorn User show, due to take place at the end of July, has been postponed until an 'unspecified date'. The reason for this sudden change is rather unclear, with neither Redwood Publishing, publishers of Acorn User, or Acorn, sponsors of the show, saying very much. However, it is widely believed that Acorn were unhappy with the flea-market nature of the show, something which has increased over the past couple of years, and would rather see a much more up-market affair complete with seminars and lectures. It is thought that Acorn User are working on such a show, but it is very unlikely that it will occur this year. While Acorn's point is probably valid, one can't help thinking that a show with lots of bargain stalls will attract more people than a more formal gathering. It remains to be seen whether another organiser, such as Database Publications, steps in to fill the gap.

ACORN PRICE INCREASES

It now seems almost certain that Acorn will announce an increase in the price of the Archimedes within the next few days. It is rumoured that this increase will be around 15%, putting about £150 on the price of an A310 with colour monitor. The price rise is attributed to increasing costs worldwide of certain integrated circuits used in the Archimedes. In particular, the price of dynamic RAM chips has increased substantially. It is not clear if the price of the Master will also rise, although any further increase in chip prices will make this almost inevitable. BEEBUG will continue to supply machines at the old price while stocks last.

SEEING DOUBLE IN VIEW

Tubelink has just released a package that allows two documents to be edited in the View wordprocessor at the same time. The package, appropriately called Double View is 32K long, and is currently available for the Master and Compact as either two 16K ROMs or as a ROM image on disc to be loaded into sideways RAM. A model B and B+ version is promised shortly. Double View works in conjunction with your existing copy of

View. The disc sells for £39.95 and the ROMs for £49.95. As well as letting you work on two documents simultaneously, Double View also offers many other improvements to View. A complete on-screen help system using pull-down menus is provided, and it is possible to cut text from a document into a clip-board, and then paste it into another document. Facilities are also provided for faster saving of part documents, and for the easy importing of spreadsheets from Viewsheet. Double View is available from Tubelink, PO Box 641, London NW9 8TF, or credit card orders on 01-205 9393.

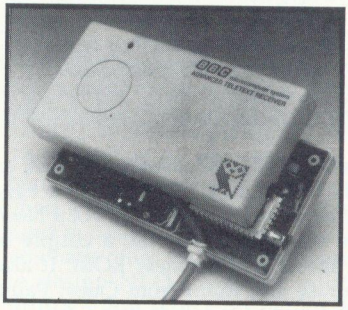
ARCHIE PIPEDREAM

Still on the subject of word processors, Colton Software have just released a version of their Pipedream combined word processor and spreadsheet for the Archimedes. Pipedream will be best known to Beeb users in the form of View Professional (reviewed in BEEBUG Vol.6 No.4), which is almost identical, although the software is also available for the IBM PC and compatibles, and is supplied as standard on the Cambridge Computer Z88. The idea of Pipedream is that of a word processor that includes the layout and functions of a spreadsheet, with some database features thrown in for good measure. The Archimedes version of Pipedream cost £113.85 including VAT, and can be obtained from Colton Software Ltd., Highcroft, The Avenue, Madingley, Cambridge CB3 8AR, or phone (0954) 210928.

ON-SITE REPAIRS

Acorn have signed an agreement with Granada Microcare to provide on-site maintenance for Archimedes systems. Granada, who are well proven in the field of computer leasing and on-site backup, promise an eight hour response to all calls, with a replacement machine being left if an immediate repair is not possible. The service contracts, which are available for the basic computer, mono or colour monitor, and any accessories, for periods from one to five years, cover all repair costs, including parts. The price of cover ranges from £94 per annum for a 300 series Archimedes to over £700 for a 440 with colour monitor over 5 years. This scheme, which should prove very attractive to business users, can be handled through BEEBUG, who will arrange cover as soon as your payment and machine details are received.

TELETEXT FROM GIS



General Information Systems, the company that brought us the Red Box home security system, has struck again, this time with a teletext adaptor for the model B and Master 128. The GIS teletext receiver comes in a very smart looking box which is the same cream colour as the computer's case and carries the BBC logo and owl. The adaptor is powered from the computer, the only connections being a lead to the user port, and a socket for a TV aerial lead. The GIS system is fully approved by the BBC, replacing the old Acorn teletext adaptor, and comes with the latest version of BBC Soft's ATS ROM. Like the rival from Morley, the GIS unit has a built-in microprocessor, but unlike the Morley system, full control of all the primitive operations is still possible. The complete system costs £149 inclusive, and can be obtained direct from GIS, Croxton Park, Croxton, Cambridgeshire PE19 4SY.

WORDWISE PLUS 2

Wordwise Plus 2, IFEL's enhanced version of Wordwise Plus, which was reviewed in BEEBUG Vol.6 No.7, has been reduced in price to just £32.95. Additionally, for a limited period, IFEL are supplying free with each Wordwise Plus 2 a fast sorting package. Existing Wordwise Plus 2 users can buy the sorting software for a nominal price by contacting IFEL. IFEL are at 36 Upland Drive, Plymouth, Devon PL6 6BD, or telephone (07555) 7286.

ConQuest

Roger Burg reports on Watford Electronics' latest enhancements to the Quest Mouse package.

Product	ConQuest
Supplier	Watford Electronics Jessa House, 250 High St, Watford, Herts, WD1 2AU. Tel. (0923) 37774
Price	£33.35 inc. VAT

Watford Electronics has released ConQuest, an add-on to its excellent mode 1 graphics package Quest Paint, reviewed in BEEBUG Vol.6 No.7. The package contains a manual, a couple of information sheets and the new ROM. It also needs the mouse and ROM from Quest Paint. Once installed, seven extra drawing features become available from within Quest. Five of these features use the Acorn Graphics ROM directly.

If you have a standard BBC model B you will need an Acorn Graphics ROM in addition to the Quest and ConQuest ROMs. A disc system is also essential. Bearing in mind the number of ROMs which need to be resident inside the machine, some kind of ROM expansion board will also be needed. It is also recommended that more memory be fitted in the form of sideways RAM if ConQuest is to be used to its full potential. Fortunately the Master meets all these requirements as supplied.

Once installed, Conquest gives the entire Quest system greater compatibility with different shadow and sideways RAM boards. On its own, it provides a font editor. You can enter the enhanced package either from Quest Paint, as before, or from ConQuest's font editor using the command *FONTEDIT or *WFONTEDIT (in the event of any ROM command clashes).

THE FONT DESIGNER

The new font designer is a first class utility. Font editors flatter a mouse more than most programs, and the precision and snappy responsiveness of this one are ideal. It has all the features which I can think of, except the ability to display a line of text, and it has a couple of new features which I haven't seen before. One of them adds or removes a complete row or column of pixels. This avoids the most befuddling and time-wasting job of balancing a font's proportions. Only the option to design Quest's brushes rather than its fonts, is less than idiot-proof.

But Quest's fonts are defined in 16 by 16 pixel grids, and spacing of characters like "i"s or "w"s is fixed in steps of 16 pixels. As the font editor necessarily adopts these restrictions, its output on screen is never more than tolerable.

NEW GRAPHICS FACILITIES

From the font editor, either *(W)PAINT or the Paint icon lead into Quest Paint. An extra "I/O" menu now presents the additional RAM and filing facilities, and further "Global FX" options are included as descriptive mouse-pointers. Five of these are called from Acorn's Graphics ROM: the ellipses, solid or filled option, arcs, sectors and segments. These are important facilities, and as I bemoaned the lack of rotatable ellipses in the earlier review, I must applaud them here! Acorn's Graphics ROM is slower than Quest and cannot use all its options of protected and cycled colours, but this is a small price to pay.

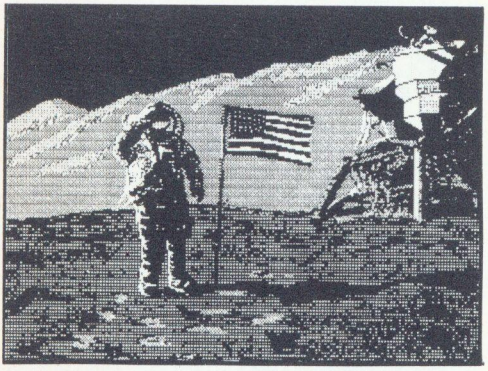
The two new features which ConQuest provides operate on the image in the cut-and-paste window. This can be read from sideways RAM or from the screen, and it is copied back to a new position, at any angle or distorted to any four-sided shape. Both are easy to use.

The implementation of the rotation is exemplary. One button press establishes the centre of the destination window, which is outlined on the screen, varying its angle of rotation smoothly according to the horizontal position of the mouse pointer. A second press can remove the point to reposition it, or begin plotting the image slowly and thoroughly.

The distortion option is similar except that drawing the initial four-sided figure is more prone to errors, and pressing the quit button undoes all the points, not just the most recent one. However, ConQuest plots the paste image

to fit the new shape equally thoroughly and fairly accurately in any four sided figure.

ConQuest does not cope with all quadrilateral distortions. However it was not intended to foreshorten "COKE" as it appears wrapped round the side of a can, nor draw a chess board receding in perspective. But both routines open up possibilities for experiment, and even if the results have to be modified by hand or rejected, Quest will do either.



THE MANUAL ON USING EXTRA RAM

If you're thinking of buying ConQuest to let Quest make use of extra RAM, or if you have more than one filing system already installed, then go to your dealer and check the manual first. It is precise and readable and describes the few incompatibility problems well. In short, the Graphics ROM, if used with ADFS, takes a little too much memory on a BBC B, and there is a problem with level three file servers. The manual also explains how to avoid problems which may arise between the Quest system and other resident firmware. Apart from these exceptions, Watford Electronics claim that the Quest system works happily on most combinations of hardware.

To ensure that ConQuest can use just about any proprietory RAM extension, a customised RAM driver routine of up to 64 bytes can be placed in memory at location &140 to select write access to your particular RAM board. While maximum compatibility is always welcome, it would not be unreasonable to expect a little more support in this respect.

The typestyle of a manual seldom causes comment, but the improvement over the Quest manual's typeface is outstanding. It was difficult to maintain confidence in a graphics package whose manual is graphically weak.

CONCLUSIONS

Unless you have a Master, you will need to purchase the Acorn's Graphics ROM in order to run ConQuest, and if you want to use its full potential, you will be advised to install extra RAM in your machine.

When using the package you will find that Quest's button presses do not provide all the options required, and ConQuest's extra functions slightly increase the complexity of this, a small consideration, but a nuisance when you are doing something tricky or important.

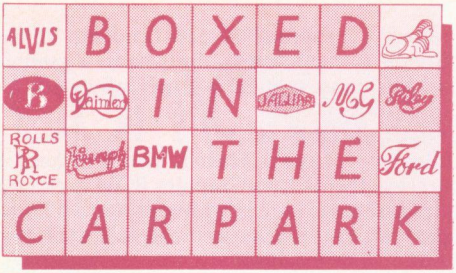
The font designer, however good, is but a small bonus in Quest. If you cannot design a dozen good letters straight onto the screen you won't need to design 127 bad ones in the editor. But if you have other applications - perhaps you want to design NLQ letters for your own printer utility - you won't do better than with this package.

The new graphics features are useful: rotated ellipses for example, and the window transformation, both have great imaginative potential. To say that they add little to Quest must be understood as a compliment to one of the best graphics programs about, and not a slight on ConQuest.

Quest was not written with perspective in mind. You have to work hard at it to convey depth, and this is not seriously improved by the new features. It is also intended as a mode 1 designer, and apparently writes straight to the screen, which probably accounts for its speed and reduces the code, but consequently it will not use spooled files, second processors or other screen modes. So ConQuest does not do everything! But then as an add-on it only costs £33.35, and if you use mode 1 graphics seriously, and need those useful rotated ellipses - treat yourself.

NOTE

Please note that ConQuest does not work with very early versions of Quest. Should you have any compatibility problems, Watford Electronics will update your existing Quest ROM to the latest version for a small handling charge. B



This is one of those maddeningly simple yet frustrating games that rapidly becomes addictive. In this case G.N.Steeper's version is based on an original game over 40 years old.

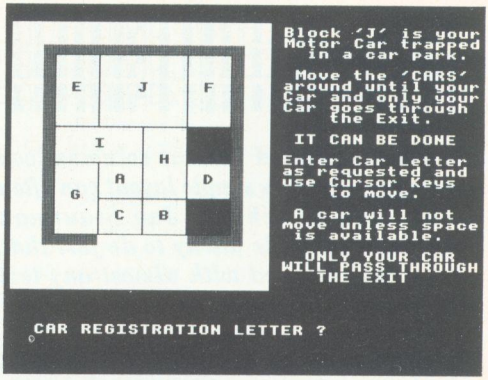
This program is an implementation on a moving-block type of puzzle that came into my possession over 40 years ago, well before the days of the microcomputer. The original was made from cardboard, but I made a more durable copy from perspex. This has stood the test of time and came to light again just recently when I was tidying up. I decided to have a go at producing a computer version of the puzzle, and the following listing is the result.

In outline, the object of the game is to 'shuffle' cars around in a carpark so that you may extricate your own vehicle (parking problems are clearly not new). In the original puzzle, the other vehicles were labelled Rover, Austin, Hillman, Riley and the like, which now arouses a touch of nostalgia for these old marques. The puzzle, though, is just as tantalising.

Type the program in and save it. When run, the program is quite self-explanatory, as the operating instructions are always in view. Do take special care to get the spacing right when typing in lines 1020 to 1070 as any error here spoils the presentation of the instructions. Moving cars around the carpark is no more complicated than entering the car's registration letter (in upper case) followed by the cursor key to indicate the direction of movement. When you have given the puzzle a fair trial and begin to doubt your ability to solve it, just press Escape and the solution will be revealed. Pressing Shift-Escape will exit from the program. Have fun.

```
   10 REM Program CarPark
   20 REM Version B1.0
   30 REM Author  G.N.Steeper
   40 REM BEEBUG  May 1988
   50 REM Program subject to copyright
   60 :
  100 finish=FALSE:error=FALSE
  110 ON ERROR PROCerror: IF finish THEN
 END
  120 IF NOT error MODE 1: PROCmenu
  130 error=FALSE
  140 REPEAT
  150 *FX4,1
  160 IF JY%=468 PROCEND
  170 IF Solve=1 READ M$:IF M$="X" PROCE
ND
  180 IF Solve=1 READ Action
  190 IF Solve<>1 REPEAT:PRINT TAB(2,29)
"CAR REGISTRATION LETTER ?  ";:M$=GET$:U
NTIL M$>="A" OR M$<="J"
  200 PRINTTAB(28,29) M$;
  210 IF M$="A" X%=AX%:Y%=AY%:W=a:D=b:x=
c:y=e:PROCmove:AX%=X%:AY%=Y%
  220 IF M$="B" X%=BX%:Y%=BY%:W=a:D=b:x=
c:y=e:PROCmove:BX%=X%:BY%=Y%
  230 IF M$="C" X%=CX%:Y%=CY%:W=a:D=b:x=
c:y=e:PROCmove:CX%=X%:CY%=Y%
  240 IF M$="D" X%=DX%:Y%=DY%:W=a:D=b:x=
c:y=e:PROCmove:DX%=X%:DY%=Y%
  250 IF M$="E" X%=EX%:Y%=EY%:W=a:D=B:x=
c:y=E:PROCmove:EX%=X%:EY%=Y%
  260 IF M$="F" X%=FX%:Y%=FY%:W=a:D=B:x=
c:y=E:PROCmove:FX%=X%:FY%=Y%
  270 IF M$="G" X%=GX%:Y%=GY%:W=a:D=B:x=
c:y=E:PROCmove:GX%=X%:GY%=Y%
  280 IF M$="H" X%=HX%:Y%=HY%:W=a:D=B:x=
c:y=E:PROCmove:HX%=X%:HY%=Y%
  290 IF M$="I" X%=IX%:Y%=IY%:W=A:D=b:x=
C:y=e:PROCmove:IX%=X%:IY%=Y%
  300 IF M$="J" X%=JX%:Y%=JY%:W=A:D=B:x=
C:y=E:PROCmove:JX%=X%:JY%=Y%
  310 UNTIL FALSE
  320 :
 1000 DEF PROCmenu
 1010 VDU28,23,28,39,1
 1020 PRINT"Block 'J' is yourMotor Car t
rapped  in a car park."
 1030 PRINT'" Move the 'CARS' around unt
il yourCar and only yourCar goes through
     the Exit."
 1040 PRINT'" IT CAN BE DONE"
 1050 PRINT'"Enter Car Letter as request
ed and use Cursor Keys      to move."
 1060 PRINT'" A car will not  move unles
s space is available."
```

```
 1070 PRINT'"  ONLY YOUR CAR  WILL PASS
THROUGH   THE EXIT"
 1080 VDU26
 1090 PROCsetup
 1100 Solve=0
 1110 ENDPROC
 1120 :
 1130 DEF PROCmove
 1140 PROCcheck
 1150 PROCclear
 1160 IF Solve<>1 Action=GET
 1170 IF Action=139 AND GOU=0 PROCdraw(0
,M$,X%,Y%,W,D,x,y):Y%=Y%+108:PROCdraw(2,
M$,X%,Y%,W,D,x,y):ENDPROC
 1180 IF Action=138 AND GOD=0 PROCdraw(0
,M$,X%,Y%,W,D,x,y):Y%=Y%-108:PROCdraw(2,
M$,X%,Y%,W,D,x,y):ENDPROC
 1190 IF Action=137 AND GOR=0 PROCdraw(0
,M$,X%,Y%,W,D,x,y):X%=X%+120:PROCdraw(2,
M$,X%,Y%,W,D,x,y):ENDPROC
 1200 IF Action=136 AND GOL=0 PROCdraw(0
,M$,X%,Y%,W,D,x,y):X%=X%-120:PROCdraw(2,
M$,X%,Y%,W,D,x,y)
 1210 ENDPROC
 1220 :
 1230 DEFPROCcheck
 1240 f=16:g=50:h=150
 1250 U1=POINT(X%+g,Y%+f):U2=POINT(X%+h,
Y%+f):IF U1=0 AND U2=0 Col=0
 1260 D1=POINT(X%+g,Y%-D-g):D2=POINT(X%+
h,Y%-D-f):IF D1=0 AND D2=0 Col=0
 1270 L1=POINT(X%-f,Y%-g):L2=POINT(X%-f,
Y%-h):IF L1=0 AND L2=0 Col=0
 1280 R1=POINT(X%+W+f,Y%-g):R2=POINT(X%+
W+f,Y%-h):IF R1=0 AND R2=0 Col=0
 1290 ENDPROC
 1300 :
 1310 DEFPROCdraw(Col,S$,X%,Y%,W,D,x,y)
 1320 GCOL0,Col
 1330 MOVE X%,Y%
 1340 PLOT 69,X%+W,Y%
 1350 PLOT 85,X%,Y%-D
 1360 PLOT 85,X%+W,Y%-D
 1370 VDU5:MOVEX%+x,Y%-y:GCOL0,0:PRINTS$
:VDU4
 1380 ENDPROC
 1390 :
 1400 DEF PROCerror
 1410 IF ERR=17 AND INKEY-1 THEN CLS:PRI
NT"Escape pressed at line ";ERL'':finish
=TRUE
 1420 IF ERR=17 AND NOT(INKEY-1) THEN So
lve=1:PROCsetup:PRINTTAB(2,29)"SOLUTION
COMING UP"SPC(15):VDU7:MELA=INKEY(300):V
DU7:error=TRUE
```




```
 1430 IF ERR<>17 THEN CLS:REPORT:PRINT ;
" at line ";ERL'':finish=TRUE
 1440 ENDPROC
 1450 :
 1460 DEFPROCsetup
 1470 PROCdraw(3,"",0,1000,700,800,0,0)
 1480 PROCdraw(1,"",84,932,544,600,0,0)
 1490 PROCdraw(0,"",116,904,480,540,0,0)
 1500 PROCdraw(3,"",236,360,240,28,0,0)
 1510 A=232:a=112:B=208:b=100
 1520 C=104:c=40:E=80:e=32
 1530 EX%=120:EY%=900
 1540 PROCdraw(2,"E",EX%,EY%,a,B,c,E)
 1550 JX%=240:JY%=900
 1560 PROCdraw(2,"J",JX%,JY%,A,B,b,E)
 1570 FX%=480:FY%=900
 1580 PROCdraw(2,"F",FX%,FY%,a,B,c,E)
 1590 GX%=120:GY%=684
 1600 PROCdraw(2,"G",GX%,GY%,a,B,c,E)
 1610 IX%=240:IY%=684
 1620 PROCdraw(2,"I",IX%,IY%,A,b,C,e)
 1630 HX%=480:HY%=684
 1640 PROCdraw(2,"H",HX%,HY%,a,B,c,E)
 1650 AX%=240:AY%=576
 1660 PROCdraw(2,"A",AX%,AY%,a,b,c,e)
 1670 BX%=360:BY%=576
 1680 PROCdraw(2,"B",BX%,BY%,a,b,c,e)
 1690 CX%=120:CY%=468
 1700 PROCdraw(2,"C",CX%,CY%,a,b,c,e)
 1710 DX%=480:DY%=468
 1720 PROCdraw(2,"D",DX%,DY%,a,b,c,e)
 1730 ENDPROC
 1740 :
 1750 DEFPROCclear
 1760 GOU=1:GOD=1:GOL=1:GOR=1
 1770 IF W=a AND U1=0 GOU=0
 1780 IF W=A AND U1=0 AND U2=0 GOU=0
 1790 IF W=a AND D1=0 GOD=0
```

Continued on page 29

MULTI-COLUMN PRINTING

Documents printed in neat columns look really good, yet this simple layout can often be difficult to achieve. Jan Stuurman provides a versatile utility to do just that, and it may be used with almost any text file, whatever its source.

The utility presented in this article formats any (spooled) text file into columns, and prints the result using an Epson FX-80 or compatible printer. Start by typing in the program, taking extra care with the assembler sections, and save it.

```
      MULTI-COLUMN PAGE PRINTER

Enter source filename      MULTCOL
Enter print mode (E/P/C) C
 (Elite/Pica/Condensed)
Print mode: 4  Characters/line: 132
Enter number of columns    4
Space between columns      5
Column width (characters): 29
Page length (<174)         60
```

When you run the program, it will ask for the name of the source (text) file to be formatted. The program then offers a choice of three print modes: 80 characters per line (cpl) Elite, 96 cpl Pica, or 132 cpl Condensed. You must then specify the number of columns, the space between them (in characters) and the number of lines per page to be printed. All text lines output are left-justified, and are split at word boundaries except when the word is longer than the width of the column. A Return in the text signals the end of a line, while the '{' character may be used to indicate the end of a column, and similarly the '}' character to mark the end of a page, although these two special characters can be changed if desired by altering lines 3570 and 3580. When the program has finished printing a page it will beep and wait for the space bar to be pressed, allowing the paper to be changed if printing on individual sheets. If you just want to print continuous sheets without having to press space each time then remove line 180 from the listing.

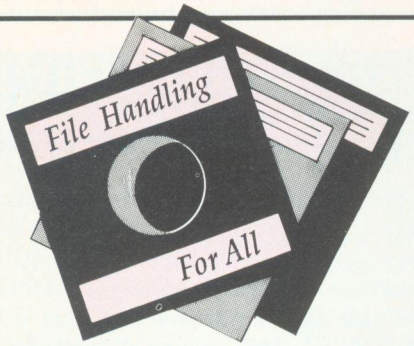
The only Epson-specific code is the setting of the print mode in line 150, and the re-setting of the printer in line 200. The print mode is selected in lines 2070-2120. It would not be difficult to change the program to work with any other printer that supports the print modes used, but which uses different codes to select them.

Almost any text file may be processed by this utility, whether originated through a word processor, text editor or any other means. Just make sure that the text is in pure ASCII format first (i.e. spool the text out from a word processor such as Wordwise, or remove any formatting commands as in View), and away you go.

```
   10 REM Program MultCol
   20 REM Version B1.0
   30 REM Author  Jan Stuurman
   40 REM BEEBUG  May 1988
   50 REM Program subject to copyright
   60 :
  100 MODE7:HIMEM=PAGE+&1400
  110 ON ERROR GOTO900
  120 PROCtitle
  130 PROCinput
  140 PROCassem
  150 VDU2,1,27,1,33,1,pmode%,3
  160 VDU28,0,24,39,22
  170 REPEAT CLS:VDU2:CALL prtpage:VDU1,
13,3,7:*FX15,1
  180 IF ?eofflg=0 PRINTTAB(4)CHR$133"Pr
ess <SPACE> to continue...":IF GET=32
  190 UNTIL ?eofflg=&FF
  200 CLOSE#S%:VDU2,1,27,1,64,3,26
  210 END
  220 :
  900 ON ERROR OFF
  910 CLOSE#0:VDU3
  920 REPORT:PRINT" at line ";ERL
  930 END
```

```
  940 :
 1000 DEFPROCtitle
 1010 FOR I=0 TO 1:PRINTTAB(3,I)CHR$141C
HR$129CHR$157CHR$131"MULTI-COLUMN PAGE P
RINTER  "CHR$156:NEXT
 1020 ENDPROC
 1030 :
 2000 DEFPROCinput LOCALI%
 2010 VDU28,0,24,39,5:REPEAT CLS
 2020 PRINTCHR$130"Enter source filename
";TAB(25)CHR$131;
 2030 INPUT""sf$:S%=OPENINsf$
 2040 IF S%=0 VDU7:PRINTTAB(10)CHR$129"N
O SUCH FILE":I%=INKEY250
 2050 UNTIL S%<>0
 2060 VDU28,0,24,39,7
 2070 REPEAT CLS:PRINTCHR$130"Enter prin
t mode (E/P/C)"
 2080 PRINTCHR$130" (Elite/Pica/Condense
d)";TAB(25,0)CHR$131;
 2090 P%=GET AND &5F
 2100 UNTIL P%=69 OR P%=80 OR P%=67:PRIN
T CHR$P%':pmode%=-(P%=80)-4*(P%=67)
 2110 mlin%=80-16*(P%=80)-52*(P%=67)
 2120 PRINT'CHR$134"Print mode:"CHR$133;
pmode%;TAB(15)CHR$134"Characters/line:"
CHR$133;mlin%
 2130 VDU28,0,24,39,12
 2140 REPEAT CLS:PRINT CHR$130"Enter num
ber of columns";TAB(25)CHR$131;
 2150 INPUT""ncol%
 2160 UNTIL ncol%>0ANDncol%<=mlin%
 2170 VDU28,0,24,39,14
 2180 REPEAT CLS:PRINTCHR$130"Space betw
een columns";TAB(25)CHR$131;
 2190 INPUT""cspc%
 2200 UNTIL cspc%>0 AND cspc%<=mlin%/(nc
ol%-1)
 2210 cwid%=(mlin%-(ncol%-1)*cspc%)/ncol
%
 2220 PRINT'CHR$134"Column width (charac
ters):"CHR$133;cwid%
 2230 mpage%=(&7C00-HIMEM)/mlin%
 2240 VDU28,0,24,39,18
 2250 REPEAT CLS:PRINTCHR$130"Page lengt
h (<";mpage%;")";TAB(25)CHR$131;
 2260 INPUT""plen%
 2270 UNTIL plen%>0ANDplen%<=mpage%
 2280 ENDPROC
 2290 :
 3000 DEFPROCassem
 3010 base=&70:lbase=&72
 3020 bufptr=&74:?&74=0
 3030 eofflg=&75:?&75=0
 3040 eopflg=&76:eocflg=&77:eolflg=&78
 3050 col=&79:lin=&7A
 3060 off=&7B:?&7B=cwid%+cspc%
 3070 buffer=&67B
 3080 osbget=&FFD7:oswrch=&FFEE
 3090 :
 3100 FOR pass=0 TO 2 STEP 2:P%=&900
 3110 [OPTpass
 3120 .prtpage LDA #0:STA eopflg
 3130 LDA #HIMEM MOD256:
STA base:STA lbase
 3140 LDA #HIMEM DIV256:
STA base+1:STA lbase+1
 3150 JSR clearpage
 3160 LDA #ncol%:STA col
 3170 .columnloop LDA #0:STA eocflg
 3180 LDA base:STA lbase:
LDA base+1:STA lbase+1
 3190 LDA #plen%:STA lin
 3200 .lineloop LDA #0:STA eolflg
 3210 JSR makeline
 3220 .chkeol LDA eolflg:BNE chkeoc
 3230 JSR bufferreset
 3240 .chkeoc LDA eocflg:BNE chkeop
 3250 CLC:LDA lbase:ADC #mlin%:STA lbase
 3260 BCC nocarry:INC lbase+1
 3270 .nocarry DEC lin:BNE lineloop
 3280 .chkeop LDA eopflg:BNE print
 3290 CLC:LDA base:ADC off:STA base
 3300 BCC noc:INC base+1
 3310 .noc DEC col:BNE columnloop
 3320 .print
 3330 LDA #HIMEM MOD256:STA base
 3340 LDA #HIMEM DIV256:STA base+1
 3350 LDX #plen%
 3360 .prloop1 LDY #0
 3370 .prloop2 LDA #1:JSR oswrch
 3380 LDA (base),Y:JSR oswrch
 3390 INY:CPY #mlin%:BNE prloop2
 3400 CLC:LDA base:ADC #mlin%:STA base
 3410 BCC pnoc:INC base+1
 3420 .pnoc DEX:BNE prloop1
 3430 RTS
 3440 :
 3450 .clearpage
 3460 LDX #plen%
 3470 .cloop1 LDY #0:LDA #&20
 3480 .cloop2 STA (lbase),Y
 3490 INY:CPY #mlin%:BNE cloop2
 3500 CLC:LDA lbase:ADC #mlin%:STA lbase
 3510 BCC cnoc:INC lbase+1
```

Continued on page 64



Mike Williams and David Spencer commence a major series of articles on the considerable topic of file handling in Basic.

The ability to store data for subsequent access and manipulation is one of the most important and commonplace functions of a computer system, and learning how to write programs to achieve this is often a major milestone for anyone developing their programming skills. Indeed, we feel sure that there are many Basic programmers who even now still feel that this subject is too daunting to be mastered. This series of articles will start right from the beginning, so no one should feel that they are excluded.

But these articles are not aimed just at the beginner. The series will develop a general understanding of file structures and file handling techniques which we hope will prove equally useful to experienced programmers.

CARD INDEX FILES

This month we will start from scratch by defining some useful terms, and learning some basic techniques. The simplest form of file is probably one based on the familiar card index. Such a file consists of a number of *records* (or cards), one for each entry in the file, while each record is divided into several separate pieces of information called *fields*. In a card index, and indeed in most files, every record in a given file follows exactly the same format.

Whenever you are going to create a data file, one of the first things you need to do is to sit down and decide just what fields each record is going to contain. It is worth spending some time on this, whatever type of file organisation is going to be used, as changing the file structure (that is the organisation of the file) at a later stage can be difficult if not impossible.

So let's take an example. We will create a file to contain the names and addresses of friends with details of their birthdays. Fairly obviously we need to store the following information about each person in the file:

FIELD	VARIABLE NAME
Name	Name$
Address	Address$
Telephone Number	Phone$
Birthday	Date$

Since we will need to refer to these pieces of information (or fields) within any program we will use **Name$** and **Address$** as variable names for the first two items, but for brevity **Phone$** and **Date$** for the last two respectively. All four fields are here treated as strings.

> **BASIC I**
> Users of Basic I should note that any occurrence of OPENUP in programs listed in this series should be replaced by OPENIN. This is discussed in more detail in this first article.

CREATING A DATA FILE

We are now ready to write a simple program to create a file containing this information. The file handling instructions needed will be explained using this example.

```
Program:CREATE
 100 MODE 3
 110 ON ERROR GOTO 220
 120 PRINTTAB(10,1)"CREATE DATES FILE"
 130 VDU28,0,24,79,3
 140 F=OPENOUT("DATES")
 150 REPEAT
 160 INPUTLINE'"Name: " Name$
 170 INPUTLINE"Address: " Address$
 180 INPUTLINE"Phone number: " Phone$
 190 INPUTLINE"Birthday: " Date$
 200 PRINT#F,Name$,Address$,Phone$,Date$
 210 UNTIL FALSE
 220 IF ERR=17 THEN PRINT'"File DATES C
reated OK"
 230 IF ERR<>17 THEN REPORT:PRINT" at li
ne ";ERL
 240 CLOSE#F:VDU26
 250 END
```

The program selects mode 3 and displays a title at the top of the screen before defining a text window. This will contain the dialogue which ensues as we enter the data, without corrupting or losing our main heading.

The first file related instruction is in line 140. Before a file can be used it must be *opened* ready for use. In this instance (line 140) we use OPENOUT (for output from computer to disc), specifying the name by which the file is to be known (DATES). OPENOUT checks to see if a file of the name specified already exists, and if so deletes it, before creating and opening a file ready for our use. Because of this, OPENOUT should only be used when you want to create a new file.

```
        DISPLAY DATES FILE

Name: Jane Smith
Address: 2 Long Lane, Harrow, Middlesex.
Phone number: 01-233 5544
Birthday: 5th May

Name: John Brown
Address: 3 Hyacinth Drive, Stevenage, Herts.
Phone number: 44665
Birthday: 24th March

Name: Jenny White
Address: The Old Barn, Manor Lane, Risely, Bedfordshire.
Phone number: 0234-55677
Birthday: 26th September

Name: Jacky Bailey
Address: 25 Acacia Avenue, Frendlesham, Near Norwich.
Phone number: 0603-55678
Birthday: 10th June
```

When a file is opened, Basic links it to the program via a *channel*, and the number of this channel (called a *channel number* or *handle*) is returned by OPENOUT, and in our program assigned to the variable F. Any variable could be used here, but from now on all references to the file must use the file handle rather than the file name.

The REPEAT-UNTIL loop (lines 150 to 210) prompts for the input of data for each record and writes it to the file. The use of INPUTLINE means that data typed in may contain commas, quotes and the like. The PRINT# instruction in line 200 is the one which actually sends a complete record to the file. This uses a modified form of the normal PRINT statement. The difference is the reference to the channel number. We could have written each piece of data (field) to the file individually, but there is no real advantage in doing that. It also helps, for the future, to think even now in terms of reading and writing complete records.

In any case, the PRINT#F does not actually send data directly to the disc. Instead, the DFS (or ADFS) reserves part of its private memory as a *buffer*, and places the data in this. When the buffer is full the filing system copies its contents to the disc. Therefore, depending on just how much data you enter for each record, you may or may not hear the disc drive operate.

We must have some way of terminating the REPEAT-UNTIL loop. One solution is to use some special character or characters to be entered in response to the **Name** prompt, but unless we were to alter the program substantially we would still have to continue and enter dummy information (or just press Return) for the other fields as well. In this short program we have used Escape to get out of the loop, and as we have used no procedures or functions there are no problems. Once all the records have been entered, the program must close the file using a CLOSE instruction (referencing again just the channel number). It is also worth pointing out that CLOSE#0 will close all open files, and is useful in immediate mode for closing any files accidentally left open. The VDU26 at the end of the program simply restores the text window to full screen size.

READING A DATA FILE

Having created our data file, the obvious thing to do is to write another short program which will allow us to display or print the contents of our file. Here is the program to do it.

```
Program:DISPLAY
 100 MODE 3
 110 PRINTTAB(10,1)"DISPLAY DATES FILE"
 120 VDU28,0,24,79,3
 130 F=OPENIN("DATES")
 140 REPEAT
 150 INPUT#F,Name$,Address$,Phone$,Date$
 160 PRINT'"Name: ";Name$
 170 PRINT"Address: ";Address$
 180 PRINT"Phone number: ";Phone$
 190 PRINT"Birthday: ";Date$
 200 UNTIL EOF#F
 210 CLOSE#F:VDU26
 220 END
```

As you can see, the program we need to read and display the records in the file is very similar to the one we used to create it in the first place. We must open the file before we can read any records from it. This time we use OPENIN - input from file (OPENOUT would delete the existing file and create a new empty file). We have a similar REPEAT-UNTIL loop, but we must now read each record from the file before we can display its contents on the screen. Reading data from a file uses a variation on INPUT just as writing to a file uses a variation on PRINT, again referencing the channel number.

The major difference concerns the way in which the REPEAT-UNTIL loop is terminated. We do not know how many records there are, so we must just read in records until we reach the end of the file. Fortunately there is a special function in Basic to help us with this, the keyword EOF. This references the channel number, as do all file handling instructions, and returns a value of TRUE or FALSE depending upon whether the end of the file has been reached or not. Once all the records have been read and displayed the file is closed.

SIMPLE FILE UPDATING

So far so good, and if all this is new to you then we hope that you have been pleasantly surprised at how easy file handling can be. Let's proceed further. The two programs we have written so far are fine, but there is one major drawback. There is no way we can add additional records to an existing file. Every time we run the first program the existing file will be deleted and a new one created. We will now produce another program which allows an existing file to have further records added to it. Here it is.

```
Program:UPDATE
 100 MODE 3
 110 PRINTTAB(10,1)"UPDATE DATES FILE"
 120 VDU28,0,24,79,3
 130 F=OPENUP("DATES")
 140 REPEAT
 150 INPUT#F,Name$,Address$,Phone$,Date$
 160 UNTIL EOF#F
 170 ON ERROR GOTO 250
 180 REPEAT
 190 INPUTLINE'"Name: " Name$
 200 INPUTLINE"Address: " Address$
 210 INPUTLINE"Phone number: " Phone$
 220 INPUTLINE"Birthday: " Date$
 230 PRINT#F,Name$,Address$,Phone$,Date$
 240 UNTIL FALSE
 250 IF ERR=17 THEN PRINT'"File DATES Up
dated OK"
 260 IF ERR<>17 THEN REPORT:PRINT" at li
ne ";ERL
 270 CLOSE#F:VDU26
 280 END
```

As you can see, this program is largely just an amalgamation of our previous two programs. In order to add new records to our existing file, we need to find the end of that file. Because we do not know how many records already exist, the only way to do this is to start at the beginning of the file and read through all the records until we get to the end (this is like the second program but without displaying the record contents), and then continue with the equivalent of the first program to add additional records as required.

You can think of this process in terms of a pointer. When a file is opened, a pointer is placed at the beginning of the file. As records are added to a file (or read from a file) the pointer is moved forward through the file. Depending upon whether we are reading or writing, the pointer will indicate either the start of the next record to be read, or the position to start writing the next record. The concept of a pointer is important in file handling, and in future articles we will look at ways of controlling its position more directly.

OPENING FILES

You should be able to follow our latest program without too much difficulty. The one important difference is that the file is opened using OPENUP - open file for updating (not OPENIN or OPENOUT). Now much confusion seems to surround these three statements so we will try and clarify matters once and for all. According to the various user guides the three file opening instructions perform the following functions:

OPENOUT - Open for output to file only.
If the file does not exist, a new one is created. If a file with the same name exists it is deleted first and a new one created.

OPENIN - Open for input from file only.
If the file does not exist a zero channel number is returned.

OPENUP - Open a file for input and output.
If the file does not exist a zero channel number is returned. The one real source of confusion is that OPENUP does not exist in Basic I, but the action of OPENIN in Basic I is identical to OPENUP in Basic II. As a result, Basic I users should always replace OPENUP with OPENIN (in our third program for example).

When you want to create a new data file use OPENOUT, but as any existing file with the same name will be automatically deleted it may be worth checking first. For example, in our first program we might replace line 140 with:

```
  140 IF FNcheck("DATES") THEN F=OPENOUT
("DATES") ELSE VDU26:END
 1000 DEF FNcheck(filename$)
 1010 LOCAL ans$,flag%,F:flag%=FALSE
 1020 F=OPENIN(filename$)
 1030 IF F=0 THEN flag%=TRUE ELSE INPUT"
File already exists - replace (Y/N): " a
ns$:flag%=(ans$="Y"):CLOSE#F
 1040 =flag%
```

This function is quite useful and so it has been written for use with any file name which can be specified as a parameter. What the function does is to attempt to open the specified file for input from disc. If the file already exists a non-zero channel number will be allocated. This is detected by the function and the user is asked whether or not this file should be replaced. The variable flag% is set TRUE or FALSE as a result. If, when OPENIN is called, a channel number of zero is obtained, then no file of that name exists.

When accessing an existing file, we have a choice of either OPENIN or OPENUP. often, you can use OPENUP all the time for this purpose, but you should be aware of OPENIN as its more limiting functions can actually prove useful on occasion. Basic I users should use OPENIN whenever OPENUP is specified in Basic II - there is no choice here anyway.

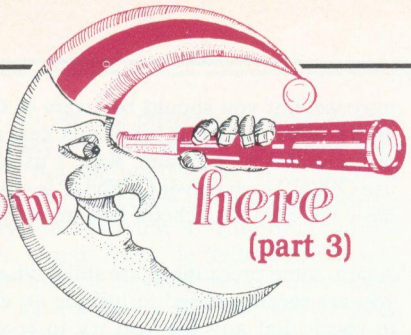
Again, some precautions are still advisable, and you are recommended to include an extra line to check that any file you try to access does exist (you might just have the wrong disc inserted for example). Simply check that the channel number is non zero. In the second and third programs we could add:

```
135 IF F=0 THEN PRINT"File not found":
    VDU26:END
```

We have now reached the point where we have three simple but complete file handling programs, one to create a file, one to display the contents of the file, and a third to update the file by adding new records to it. There are, if you think about it though, two rather obvious omissions in what we have so far achieved. There is no facility to remove a record from the file, nor is there any means of changing or modifying the contents of any record in the file, either to correct any mistake we may have made on entering the data, or just because the data has changed (change of address for example).

Unfortunately, neither of these requirements is as easy to implement as the functions we have already programmed. We need to be able to locate any record that we may wish to change, and to ensure that any amended data is written back to the file so as to replace the original record. Record deletion is no easier, because even if we determine a way to blank out any record we will be left with 'holes' in our data file which may well trip up the Display or Update programs.

We'll tackle both of these problems in the next issue of BEEBUG, where we will also begin to examine how we can make all our file handling much more general so that our programs are not tied to a particular data file as here. For now, happy filing.



Last month's article touched on C's array handling facilities and illustrated some of the concepts with a "limerick-processor". This month, as promised, we take a look at file handling with more about arrays.

Just about the only drawback with the BEEBUG C package is the lack of an editor. This is OK if you have a text or word processor, but is a bit limiting if you don't. I have designed a simple line editor that works much like the Basic editor, and embodies the concepts I want to cover this month. It is, however, quite lengthy and we'll have to cover it in two chunks, part this month and part next. The major part, which is quite long itself, appears at the end of the article.

FILES

All input and output between a C program and the real world is done via *streams*. The keyboard is an *input stream*, the VDU screen and printer *output streams*. Other streams such as files are easily established and used much as with BBC Basic. Let's open a file:

```
stream_in = fopen("c.welcome","r");
```

This is not a million miles from:

```
A=OPENIN("file")
```

Of course that's not the whole story. **stream_in** is a user-defined variable of type **FILE**. **fopen** is a standard function supplied with C. Its arguments are two strings - the first is the filename, the second the access mode. This may be one of "r" (read), "w" (write), or "a" (append), with or without a trailing "+" which signifies *update*, allowing read and write operations (similar to OPENUP in Basic II). The function returns a value which is used throughout the program to specify the stream. If the open is unsuccessful (e.g. file not found, catalogue full, etc.), a value of NULL is returned.

Naturally enough there are functions for reading from and writing to these streams. These are not part of the language as such, but are supplied in the standard library. Here is a short program to copy a file to the end of another file:

```
/** C.FAPPEND **/
/** Copies MYFILE to the end of OUTFILE  **/
#include <stdio>
#include <string>
main()
{
FILE *in, *out;
int c;
char infile [] = "MYFILE";
char outfile[] = "OUTFILE";
   /* open both files */
 if((in=fopen(infile,"r")) &
(out=fopen(outfile,"a"))){
/** if successful, do the copy **/
     while((c=fgetc(in)) != EOF)
        fputc(c,out);
     fclose(in);
     fclose(out);
   }
   else
      printf("It's all gone wrong!"); }
```

Here we have defined two pointers ***in** and ***out**, for use as file streams. We have also defined and initialised two character arrays, **infile** and **outfile** - the compiler calculates the appropriate size for the array from the initial strings. An integer **c** is used to hold the characters read because the **EOF** (End Of File) value is -1, and a variable of type **char** can never be negative.

The files are opened during the evaluation of the **if** condition. If the output file exists the file pointer is set to the end of the file. A null value returned by either occurrence of **fopen()** (they are connected by '&', the logical AND) causes the program to terminate via the else statement, otherwise the copy proceeds with the **while** loop. If the output file doesn't exist it will be created. Copying stops when the function **fgetc(in)** returns **EOF**, and the files are closed.

We have used three other file access functions; **fgetc(stream)** to read the next character, **fputc(integer, stream)** to write it and

fclose(stream). There are others which will handle strings and formatted input and output, but let's walk before we run!

MORE ABOUT ARRAYS

Last month's article dealt with the basic ideas about arrays and pointers. This month's project demands an extension of these ideas.

As we said, an array of variable-length strings is best held as a single character array with an array of pointers to the beginning of each string. For our editor, we must be able to insert text. This can be done by adding new lines to the end of the text array, but we need more information about each string in the array - we must identify the next and previous strings so that we can control the sequence of lines. To preserve the analogy with the Basic line editor, we will also number the lines (in tens, arbitrarily). The information we require about each line in our text file is:

- a pointer to this line in the text array;
- the line's "number";
- a pointer to the information about the previous line;
- a pointer to the information about the next line.

Thankfully, there is an alternative to the rather ghastly idea of processing four separate arrays. It's called a structure, an important and useful concept in C not found at all in Basic.

A structure is a group of variables which may be manipulated as a whole. The only operations allowed on a structure are getting its address (with '&') and accessing one of its members (with '->' - of which more later). This is quite enough though. Here is our structure for the array of controlling pointers:

```
struct txtcontrol{
char *ptext;   /* pointer to text array */
int linenum;   /* the line's "number" */
struct txtcontrol *prev;
struct txtcontrol *next;
} ;
```

The structure's name is **txtcontrol**; it consists of a character pointer (to identify the start of a line in the text array), the line's number, and two pointers *to the structure itself* (this is perfectly legal and very useful). To declare our array of control data we simply write, somewhere after the definition of the structure:

```
struct txtcontrol info[1000];
```

or we could simply have written **info[1000]** between the final } and ; of the structure definition. The structure definition itself does not reserve any variable space, it simply creates a new variable type (called **txtcontrol** in our example). The declaration of **info** gives us an array of variables, each of which has the defined structure.

Loading the text array from a file is fairly straightforward - open the file using **fopen()**, copy data to a text array using **fgetc()**, replace each newline **\n** with the standard C string terminator **\0** and call **insert()** to update our control array and copy the line to the text table proper. Each line points to, and is pointed to by, its neighbours. Writing the array to a file, and editing it in store, are only a little more complex.

DRIVING THE EDITOR

We need now to start defining our "language" - the commands or 'verbs' we will recognise and act on. Firstly a general format must be laid down. We will be accepting a line from the keyboard consisting of an editor command and (possibly) data. Using **getstring()** to obtain this line will save reinventing the wheel.

For ease of programming, I have assumed that the command and the data will be separated by a space or comma, and have written a function called **split()** which divides the entered command line into two parts - pre-separator and post-separator. This enables commands of the form:

```
LOAD file
```

to be handled easily. Some commands may require more than one "argument" - think of the LIST command in Basic. Here again, **split()** can be used to separate the arguments. So our general syntax will be formalised as:

```
VERB[<separator>arg[<separator>arg...]]
```

Here, the square brackets indicate optional components of the command. A separator is either a space or a comma.

If we now turn to the insertion of lines and think of Basic, you can see that a line may be thought of as a *numeric command*! Hence if the

"verb" is numeric, the remaining text is to be inserted at a place appropriate to the numeric value of the "verb" - remembering that the line numbers do not form part of the source code. If **split()** finds that the first part of a line is numeric it returns TRUE.

Finally, we turn to the vocabulary. If our editor is to resemble Basic, we should be able to LOAD a file, SAVE a file, LIST selected lines, insert and delete lines, and (because this editor is itself a program) QUIT. Later on, we can add a few bells and whistles such as AUTO, RENUMBER, DELETE, partial SAVE and an automatic compilation-on-exit.

Each of the above commands can be written as a separate function, called from **main()**. All **main** will need to do is examine the entered line and identify which function to call. For the sake of brevity, **main()** will only recognise commands typed in full in upper case and will not output any error messages. A library function **strncmp(string1, string2, num)** is used, which compares the first num characters in each string and returns zero ("FALSE") if they match. To use this in an **if** statement which requires a TRUE in the case of a match, I have used the 'NOT' qualifier '!' (as in !=, 'not equal') before each function call.

I've already touched on LOADing a program. If you examine the code, you'll see that the named file is copied into a temporary text array, character by character. Whenever a return character is found, \0 is substituted and the function **insert()** is called. Here we see structure manipulation for the first time. The function has a pointer, **ptr**, which is of type **struct txtcontrol**. This can be set to point to any element in **info** thus:

```
struct txtcontrol *ptr, *ptr1; ... ptr =
&info[n];
```

and from then on, using **ptr** in a statement will be identical in effect to using **info[n]** (until **n** is changed!). By incrementing or decrementing **ptr** we can access the next or previous element. These two statement groups are equivalent as far as **ptr1** is concerned:

```
{
n++;
ptr1 = &info[n];
}

{
ptr++;
ptr1 = ptr;
}
```

Members of a structure are identified thus:

```
old_line = info[n].linenum;
```

so using our pointer we could say:

```
old_line = *ptr.linenum;
```

A short form of this last format exists and is used in preference:

```
old_line = ptr->linenum;
```

The operator -> consists of a hyphen followed by the 'greater than' symbol. The operation can be nested thus:

```
old_line = ptr->next->linenum
```

if **ptr->next** is a pointer to the structure of which **linenum** is a member.

Now you should be able to follow the code in the largest function, **insert()**. The incoming line number may lie outside the current range - before the first line or after the last. It may also replace the first or last line. Under these conditions only, the variables **firstpointer** and **firstline**, or **lastpointer** and **lastline** will need to be updated.

To insert a line, we scan the control data array until the right position is found. Then we must link our new line to its *previous* and *next* neighbours. If we are replacing an existing line, the old line must be unlinked and the new one joined up in its place. If the new data is a zero length string we are deleting a line, so the previous and next lines are linked together. Finally, the text is copied into the text array proper.

Turning to the LIST function, we can see how the command line is split once again to give the start and finish line numbers. These are converted from string format to numeric format by **atoj()**, a home-produced version of the library function **atoi()** - "ASCII to integer". The control array is scanned for the first specified line. You cannnot just run through **info** by incrementing a subscript if lines have been inserted or deleted, so we re-initialise our

pointer (in the third portion of the **for** statement) by setting it first to **firstpointer** and subsequently to **ptr->next**. We then print succeeding lines until the second line number is exceeded. Piece of cake really, isn't it?

The SAVE function essentially LISTs the text to a file, and the code is correspondingly similar, except that newline characters (**\n**) replace nulls (**\0**). A partial SAVE would be disgustingly easy to implement - if you remember how **split()** works (returning TRUE if the first portion is numeric), the start and finish lines should precede the filename thus:

```
SAVE 10,300 progpt1
SAVE 310,700 progpt2
```

Well, that just about wraps it up for now. Next month we'll add some extra facilities to our editor, and look into the BEEBUG C compilation and linking processes.

C you around!

> NOTE:
> For those who do not have access to a C compiler, a fully compiled version of the editor will be included on the magazine disc when part 4 of this series is published in the next issue.

```
/* elementary line editor */
/* BY D MCSWEENEY (C) 1988 */
/* Beebug C series, part 3 */

#include <stdio>
#include <string>
#define TEXTMAX 8000
#define LINEMAX 500

   /* *** external variables *** */
struct txtcontrol{
  char *ptext;
  struct txtcontrol *prev;
  int linenum;
  struct txtcontrol *next;
 } info[LINEMAX];
struct txtcontrol *firstpointer, *lastpointer;
int firstline, lastline;
FILE *in, *out;
char text[TEXTMAX], *pfree;
int inc = 10;
int lineno;
int nfs; /* next free subscript in info */
char p1[6], p2[6];
/* MAIN */
main(){
char command[5], line[74];
char inline[80];
int result, n;
initialise();
  do{
   lineno = 0;
   printf("? ");
   getstring(inline, 80);
   if(split(inline, command, line))
     lineno=atoj(command);
   if(lineno > 0)
     insert(lineno, line);
   if(!strncmp(command, "LOAD",4))
     progload(line);
   if(!strncmp(command, "SAVE",4))
     progsave(line);
   if(!strncmp(command, "LIST",4)){
     split(line,p1,p2);
     editcheck(p1,p2);
     proglist(atoj(p1),atoj(p2));
     }
   } while((strncmp(command, "QUIT",4)));
printf("That's yer lot!");
}

/* the final version of getstring */
getstring(str,max)
char str[];
int max;
{
int n=0;
int a;
while((a=getchar()) != '\n' && max-- > 0){
  if(a == '\b')
    if(n > 0)
      n--;
    else
      n = 0;
  else
      str[n++] = a;
  }
for(; n<max; n++)
  str[n] = '\0';
return(n);
}

/* SPLIT */
/* separate command/line no from
   the rest of the input line
   return TRUE if part1 numeric   */
split(string, part1, part2)
```

Continued on page 62

THE BEEBUG MINI WIMP (pt 1)

If you want to be able to use windows, icons and pointers, but can't afford to buy an Archimedes, then this short series from David James is right up your street.

I have always admired the sophistication of graphics achieved by packages such as the AMX mouse and BEEBUG's Icon Master; however, these all cost money, so I set about writing my own mode 4 window system in the form of a sideways ROM image. In this, the first of three articles, I will present the ROM image which is used to manipulate the windows and icons on the screen. Subsequent articles will provide a screen-based icon designer, which is itself an example of the use of the Mini-Wimp, and we will discuss other examples of the use and application of the Mini-Wimp star commands.

The ROM image for the Mini-Wimp is just over 2.5K long, with the rest of the sideways RAM bank being used as workspace by the program. The Mini-Wimp will work well on both a model B or a Master, but in the case of a model B, at least 16K of sideways RAM is needed. The Mini-Wimp provides three basic functions: Window handling; a pointer system; and an icon plotter, and it makes use of an AMX mouse if fitted.

ENTERING THE PROGRAM

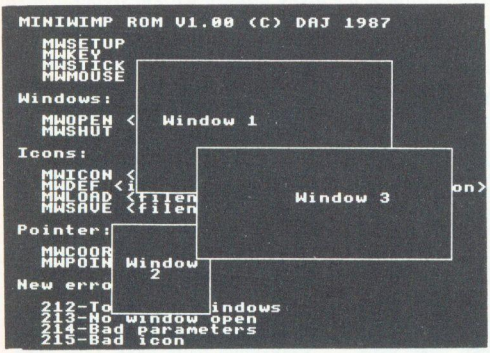
Because of the length of the original source code, we are for once publishing the Mini-Wimp in the form of a hex dump. However, the source code will be on the BEEBUG monthly disc, or you can send in an A5 SAE to get a printed listing. Entering the hex dump is made easy by the loading and error checking program in listing 1, which should be entered and saved first. Before running the loader for the first time, any file on the disc called MWROM should be deleted. When run, the loader presents the current address at the top left, which will initially be 8000. Each line of the dump should then be entered, pressing Return at the end of each line. It is not necessary to enter the spaces between the groups of characters, although these will do no harm, and both the address at the start of each line and the blank lines should be omitted. So for example, the first two lines could be entered as:

```
0000004C2B8082154AA7<Return>
004D494E4957494DCD70<Return>
```

As each line is entered, the loader checks both its length and contents, and if there is any error it will beep, print a message and prompt for the line again. If Escape or Break is pressed at any point during entry, the file will be saved up to the current line, and when the loader is next run it will detect this and start at where you left off. Once all the hex dump has been entered, the loader returns to Basic, and the file MWROM contains the final ROM image.

```
  10 REM Hex Dump Loader
  20 REM By David Spencer
  30 :
  40 ON ERROR GOTO 410
  50 *KEY10 CLOSE#X%|M*KEY 10|M
  60 READ name$,st%,end%
  70 DIM B%(9)
  80 X%=OPENUP name$:IFX%=0 X%=OPENOUT
name$
  90 PTR#X%=EXT#X%
 100 S%=(EXT#X%+st%)AND &FFF8
 110 REPEAT:REPEAT:REPEAT
 120 PRINT;~S%;":";:INPUT "" H$
 130 L$="":FORF%=1TOLENH$
 140 IFMID$(H$,F%,1)<>" " L$=L$+MID$(H$
,F%,1)
 150 NEXT
 160 IF LEN L$<>20 VDU7:PRINT"Wrong len
gth - Repeat line"
 170 UNTIL LEN(L$)=20
 180 C%=S%
 190 FOR E%=0 TO 9
 200 B%=EVAL("&"+MID$(L$,E%*2+1,2))
 210 C%=FNcrc(C%,B%):B%(E%)=B%
 220 NEXT
 230 IF C% VDU7:PRINT"Checksum error -
Repeat line"
 240 UNTIL C%=0
 250 FORF%=0TO7:BPUT#X%,B%(F%):NEXT
```

```
260 S%=S%+8
270 UNTILS%=end%
280 CLOSE#X%:*KEY 10
290 END
300 :
310 DEF FNcrc(S%,A%)
320 LOCAL F%,T%
330 S%=S%EOR A%*256
340 FORF%=1TO8
350 T%=0
360 IFS%>&7FFF S%=S%EOR&810:T%=1
370 S%=(S%*2+T%) AND &FFFF
380 NEXT
390 =S%
400 :
410 IF ERR=17 THEN CLOSE#X%:PRINT:END
420 REPORT:PRINT" at line ";ERL
430 END
440 DATA "MWROM",&8000,&8B00
```



USING THE ROM

The Mini-Wimp ROM image should be loaded in using *SRLOAD on a Master, or your normal loader on a model B, and then initialised by Control-Break. Typing *HELP will list the ROM's title, while *HELP MW will list the new star commands now available. The commands offered by the Mini-Wimp are listed below.

***MWSETUP** sets up the system and must be issued once and for all before any of the other commands.

***MWKEY** selects keyboard control of the pointer.

***MWSTICK** selects a joystick controlled pointer.

***MWMOUSE** causes the pointer commands to be passed to the AMX Super ROM, which must be installed in a higher priority socket.

***MWOPEN** <left x, bottom y, right x, top y> opens a window on the screen. The parameters are specified in character terms with the origin (0,0) being at the top left of the screen. This is the same as when setting up text windows with VDU28. The screen under the new window is saved in sideways RAM, the window border drawn, and the background cleared. You can open a maximum of three windows at a time, and each window can be up to 30 characters wide, (or 31 if one side touches the edge of the screen).

***MWSHUT** closes the last window opened, restoring the background to its former state.

***MWICON** <0 to 63> prints 1 of the 64 possible icons at the text cursor position. Each icon consists of four characters which are printed in a 2 by 2 square. The Mini-Wimp stores these definitions in memory between &5000 and &57FF. Therefore, before using icons, HIMEM should be set to &5000. The Mini-Wimp prints the icons by redefining characters 150-153.

***MWDEF** followed by an icon number and 32 further parameters define one of the 64 icons. The data is in the same form as it would be to define a normal character. This command is very cumbersome, and a complete icon designer and editor using the Mini-Wimp will be published next month.

***MWSAVE** <filename> <start icon> <end icon> saves to the named file all the icons between the given start and end numbers. For example, *MWSAVE ICONS 40 45 will save icons 40,41,42,43,44 and 45 to a file called ICONS.

***MWLOAD** <filename> <start icon> performs the opposite to *MWSAVE. The first parameter is the filename, and the second is the number of the first icon to be read in. The number of icons read depends on the length of the file. For example, *MWLOAD ICONS 1, where ICONS is the file from above, will load in icons 1 to 6.

***MWPOINTER** causes a pointer to appear on the screen. This can then be moved

around by using the cursor keys, the joystick, or an AMX mouse, depending on the option selected by *MWKEY etc. The command exits when Copy, the joystick fire button, or a mouse button is pressed, and returns the pointer's character position in the Basic variables X% and Y%. When controlling the pointer from the keyboard, there are two speeds at which the pointer moves. You can toggle between these speeds by pressing the Caps Lock key.

TESTING THE ROM

Once the ROM image has been loaded, it can be tested as follows: Type MODE 4 followed by COLOUR 129:COLOUR 0:CLS, which will select the correct mode and reverse the colours. Now type *MWSETUP to initialise all the ROM's workspace. Then fill the screen with text, using for example *HELP. A sample window can be opened by typing *MWOPEN 10,20,30,10, which should clear the middle of the screen. Finally, this window can be closed using *MWSHUT.

TECHNICAL DETAILS

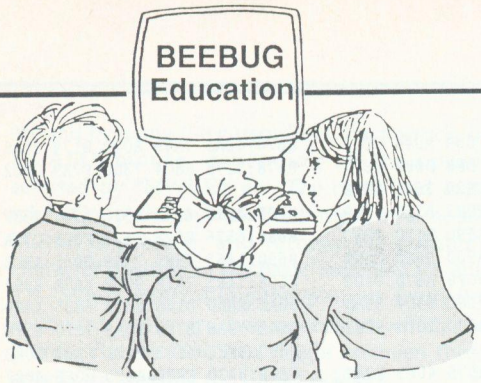
The Mini-Wimp uses memory from &70 to &8F as general workspace, and also memory from &5000 to &57FF for the icons. This icon area is just below mode 4 screen memory. Characters 150 to 153 are re-defined each time an icon is plotted, and therefore shouldn't be used elsewhere. Although the program was written for a model B with sideways RAM, it runs just as well on the Master 128 and Compact provided that shadow RAM is not used, due to the way screen memory is accessed directly.

Next month we bring you an icon designer, which not only designs icons for the Mini-Wimp, but also uses the Mini-Wimp system itself.

```
8000:0000 004C 2B80 8215 4AA7    8100:FFA9 0820 EEFF 20EE 36FD    8200:16A5 84C9 20B0 10A5 1538
8008:004D 494E 4957 494D CD70    8108:FFA9 9820 EEFF A999 C484    8208:86C9 20B0 0AA5 8438 84EF
8010:5020 524F 4D00 2843 40DF    8110:20EE FFA9 0B20 EEFF BF19    8210:E586 9003 4C1C 82A2 A88F
8018:2920 3139 3837 2044 3CC7    8118:4CA2 8086 77A2 008A 971C    8218:034C EE87 A683 CAA4 7C88
8020:6176 6964 204A 616D 934A    8120:4820 3981 B010 8578 14F7    8220:8688 A585 38E5 8318 C578
8028:6573 0008 C904 F009 85E9    8128:68AA A578 9DFB 8AE8 3B67    8228:6903 8573 A584 38E5 7724
8030:C909 D003 4C0E 8828 9A0A    8130:E477 D0EB 1860 6838 423E    8230:8618 6903 8574 A583 2DCB
8038:6048 9848 8A48 B1F2 C705    8138:6020 B181 A200 B1F2 40CE    8238:D003 E8C6 73A5 85C9 0DEA
8040:29DF C94D D055 C8B1 3A71    8140:C920 F020 C92C F01C B110    8240:27D0 02C6 73A5 86D0 D22E
8048:F229 DFC9 57D0 4CC8 9FC5    8148:C90D F018 C930 9012 7464    8248:03C8 C674 A584 C91F 4841
8050:A200 8478 A478 BD53 809A    8150:C93A B00E 38E9 3095 DD59    8250:D002 C674 A573 C921 2603
8058:8AC9 20F0 1D85 77B1 6B7E    8158:8DE8 C8E0 04F0 034C 9E05    8258:B0BD 0A0A 0A85 7320 A9FB
8060:F229 DFC5 77D0 05C8 69AF    8160:3E81 3860 C88A F0FA 6E67    8260:E783 A687 E003 D005 B492
8068:E84C 5680 8A29 F818 4762    8168:E003 F01D E002 F00D AE24    8268:A201 4CEE 87E6 878A 5AA0
8070:6908 AAC9 60F0 244C 3BF8    8170:A58D 858F A900 858D DCD6    8270:A204 182A CAD0 FBAA 0213
8078:5480 B1F2 C90D F007 21C0    8178:858E 4C89 81A5 8E85 6ED5    8278:A57B 9D2B 8BE8 A57D 8758

8080:C920 F003 4C9B 808A 9784    8180:8FA5 8D85 8EA9 0085 8FE6    8280:9D2B 8BE8 A57F 9D2B 1D41
8088:29F8 186A 6AAA BDB3 A7F8    8188:8DA9 00A6 8DF0 0818 3DBD    8288:8BE8 A581 9D2B 8BE8 F2D1
8090:8A85 77BD B48A 8578 8170    8190:6964 B0CE CAD0 F8A6 76DE    8290:A579 9D2B 8BE8 A57A 1C12
8098:6C77 0068 AA68 A868 211B    8198:8EF0 0818 690A B0C2 7827    8298:9D2B 8BE8 A575 9D2B 7E56
80A0:2860 68AA 68A8 6828 3F73    81A0:CAD0 F8A6 8FF0 0818 2F15    82A0:8BE8 A576 9D2B 8BE8 5708
80A8:A900 60A2 0120 1B81 EB6E    81A8:6901 B0B6 CAD0 F818 2FFD    82A8:A573 9D2B 8BE8 A574 C4B2
80B0:B009 A200 BDFB 8AC9 92CE    81B0:60B1 F2C9 20D0 04C8 C8E9    82B0:9D2B 8BAA A000 B175 A1EB
80B8:4090 05A2 044C EE87 AB03    81B8:4CB1 8160 A204 201B 80C7    82B8:9179 C8C4 73D0 F7A9 20D1
80C0:8575 A900 8576 A005 2CA9    81C0:81B0 54A5 8385 7BA5 5CC5    82C0:4018 6575 8575 A901 91BE
80C8:0675 2676 88D0 F9A9 BD2B    81C8:8485 7DA5 8585 7FA5 314B    82C8:6576 8576 A573 F00E 159B
80D0:5065 7685 76A2 96A0 C57B    81D0:8685 81A2 00BD FB8A E818    82D0:1865 7985 79A9 0065 786F
80D8:00A9 1720 EEFF 8A20 2212    81D8:8583 E8BD FB8A 8584 6F43    82D8:7A85 7A18 9002 E67A 5B15
80E0:EEFF B175 20EE FFC8 F923    81E0:E8BD FB8A 8585 E8BD BCF6    82E0:CAD0 D1A9 1D20 EEFF 5B21
80E8:9829 07D0 F5E8 E09A 0A60    81E8:FB8A 8586 A583 C928 B762    82E8:A900 20EE FF20 EEFF 070D
80F0:D0E7 A996 20EE FFA9 0CDC    81F0:B025 A585 C928 B01F 080F    82F0:20EE FF20 EEFF A912 2CF1
80F8:9720 EEFF A90A 20EE BD9D    81F8:A585 38E5 83C9 1FB0 6AEF    82F8:20EE FFA9 0020 EEFF E23D
```

```
8300:20EE FFA5 8385 81A9 A35F
8308:0085 8220 CF83 A581 45B1
8310:38E9 0485 7BA5 82E9 757B
8318:0085 7CA9 1F38 E584 121D
8320:8581 A900 8582 20CF 4E6D
8328:83A5 8138 E904 857D 3EBA
8330:A582 E900 857E A585 2B3C
8338:8581 E681 A900 8582 9B8D
8340:20CF 83A5 8185 7FA5 FA71
8348:8285 80A9 2038 E586 C8B7
8350:8581 A900 8582 20CF CA0C
8358:83A5 7B8D FF89 A57C F3FE
8360:8D00 8AA5 7D8D 018A 7711
8368:A57E 8D02 8AA9 048D A962
8370:FE89 20D9 83A5 818D 30AE
8378:018A A582 8D02 8AA9 A368

8380:058D FE89 20D9 83A5 6B3C
8388:7F8D FF89 A580 8D00 FB23
8390:8A20 D983 A57D 8D01 3AA6
8398:8AA5 7E8D 028A 20D9 615A
83A0:83A5 7B8D FF89 A57C 4FCB
83A8:8D00 8A20 D983 A91C DF00
83B0:20EE FFA5 8320 EEFF 2660
83B8:A584 20EE FFA5 8520 EC29
83C0:EEFF A586 20EE FFA9 EE86
83C8:0C20 EEFF 4CA2 80A2 825B
83D0:0506 8126 82CA D0F9 73C7
83D8:60A2 00BD FD89 20EE D921
83E0:FFE8 E006 D0F5 60A9 FBE3
83E8:0085 75A9 5885 7698 687D
83F0:F010 A940 1865 7585 0046
83F8:75A9 0165 7685 7688 E7A2

8400:D0F0 8AA8 F010 A908 6B3A
8408:1865 7585 75A9 0065 6EB6
8410:7685 7688 D0F0 60A9 AB36
8418:0085 8385 86A9 2785 18D3
8420:85A9 1F85 84A9 FF85 EFF0
8428:72A9 0085 8785 7085 4A92
8430:71A9 8B85 79A9 8B85 3038
8438:7A4C A280 A202 201B 1241
8440:8190 05A2 034C EE87 05F4
8448:A200 BDFB 8AC9 28B0 F6C5
8450:F285 70E8 BDFB 8AC9 1371
8458:1FB0 E885 714C A280 ACB0
8460:8677 8478 A004 B177 7949
8468:9900 0188 10F8 A200 0472
8470:A001 4CF7 FFA5 72C9 4AE5
8478:80D0 2BA2 EEA0 8920 DFCC

8480:6084 A2F3 A089 2060 2083
8488:84A2 F8A0 8920 6084 598D
8490:A200 A001 A940 20F1 D12C
8498:FFAD 0401 8570 AD05 688B
84A0:0185 714C E284 A670 7F46
84A8:A471 20E7 8320 0485 B5D1
84B0:A9CA A200 A0FF 20F4 BC23
84B8:FF8A 2910 D00A A046 7CC3
84C0:A2FF CAD0 FD88 D0F8 6817
84C8:A913 20F4 FF20 3685 076E
84D0:A670 A471 20E7 8320 19D9
84D8:0485 2054 8590 D120 4F10
84E0:3685 A000 A900 9960 C8BA
84E8:04C8 C008 D0F8 A670 04D5
84F0:8E60 04A4 718C 6404 55AD
84F8:A90F A201 A000 20F4 98E2

8500:FF4C A280 A575 48A5 E2AE
8508:7648 A000 B175 99EB E397
8510:8A19 CB8A 59DB 8A91 9F00
8518:75C8 C008 D00D A938 4A11
8520:1865 7585 75A9 0165 E70F
8528:7685 76C0 10D0 DD68 A1DE
8530:8576 6885 7560 A000 5731
8538:B9EB 8A91 75C8 C008 F644
8540:D00D A938 1865 7585 CB4B
8548:75A9 0165 7685 76C0 5CD1
8550:10D0 E560 20C9 85E0 64FC
8558:FFD0 06A5 70F0 02C6 E339
8560:7020 E285 E0FF D008 0686
8568:A570 C927 F002 E670 73A2
8570:2097 85E0 FFD0 06A5 0203
8578:71F0 02C6 7120 B085 62EE

8580:E0FF D008 A571 C91E 8B2F
8588:F002 E671 20FB 85E0 2CA5
8590:FFF0 0218 6038 60A5 C4EE
8598:72F0 05A9 C64C 1586 9A02
85A0:A980 A202 20F4 FFA2 B449
85A8:00C0 C090 02A2 FF60 5CFF
85B0:A572 F005 A9D6 4C15 1B75
85B8:86A9 80A2 0220 F4FF 57F3
85C0:A200 C041 B002 A2FF 3531
85C8:60A5 72F0 05A9 E64C 3C09
85D0:1586 A980 A201 20F4 99CE
85D8:FFA2 00C0 C090 02A2 5DB0
85E0:FF60 A572 F005 A986 53B0
85E8:4C15 86A9 80A2 0120 5584
85F0:F4FF A200 C041 B002 B028
85F8:A2FF 60A5 72F0 05A9 6466

8600:964C 1586 A980 A200 5DA4
8608:20F4 FF8A A200 2903 C013
8610:F002 A2FF 60AA A981 5258
8618:A0FF 4CF4 FFA6 87D0 204A
8620:05A2 024C EE87 C687 2A0A
8628:CA8A A204 182A CAD0 4C64
8630:FBAA BD2B 8B85 83E8 F960
8638:BD2B 8B85 84E8 BD2B 8A8E
8640:8B85 85E8 BD2B 8B85 4FF9
8648:86E8 BD2B 8B85 79E8 49C1
8650:BD2B 8B85 7AE8 BD2B 972E
8658:8B85 75E8 BD2B 8B85 7A93
8660:76E8 BD2B 8B85 73E8 16DC
8668:BD2B 8B85 74AA A579 7CF4

8670:48A5 7A48 A000 B179 55A6
8678:9175 C8C4 73D0 F7A9 0BC2

8680:4018 6575 8575 A901 433F
8688:6576 8576 A573 F00E C71A
8690:1865 7985 79A9 0065 AAEE
8698:7A85 7A18 9002 E67A 8994
86A0:CAD0 D168 857A 6885 A4F9
86A8:79A6 87F0 21A9 1C20 857B
86B0:EEFF A583 20EE FFA5 C082
86B8:8420 EEFF A585 20EE 4D22
86C0:FFA5 8620 EEFF A91E CF07
86C8:20EE FF4C A280 A91A 2FD4
86D0:20EE FF4C A280 A900 3879
86D8:8572 4CA2 80A9 FF85 CDC7
86E0:724C A280 A980 8572 05DF
86E8:4CA2 80A2 2120 1B81 9754
86F0:9005 A203 4CEE 87A2 E2C3
86F8:00BD FB8A C940 9005 FA51

8700:A204 4CEE 8785 75A9 E897
8708:0085 7620 DD87 A000 8992
8710:B9FC 8A91 75C8 C020 0FEC
8718:D0F6 4CA2 8020 B181 DE68
8720:A200 A900 9D00 01E8 44E4
8728:E012 D0F8 A200 B1F2 9CD3
8730:9D20 01E8 C8C9 0DF0 F070
8738:0BC9 20D0 F1CA A90D 07A1
8740:9D20 0160 A203 4CEE 49A1
8748:8720 1D87 A201 201B A4E6
8750:8190 05A2 034C EE87 FAFA
8758:A200 BDFB 8A85 75A9 9AFC
8760:0085 7620 DD87 A575 78E2
8768:8D02 01A5 768D 0301 C42E
8770:A920 8D00 01A9 018D B320
8778:0101 A9FF A200 A001 2BC3

8780:20DD FF4C A280 201D 76AE
8788:87A2 0220 1B81 9005 5FD0
8790:A203 4CEE 87A2 00BD CB1E
8798:FB8A 8575 A900 8576 BCEE
87A0:20DD 87A5 758D 0A01 4219
87A8:A576 8D0B 01A2 01BD 11BB
87B0:FB8A 8575 E675 A900 95E5
87B8:8576 20DD 87A5 758D 855B
87C0:0E01 A576 8D0F 01A9 C7AB
87C8:208D 0001 A901 8D01 1C3F
87D0:01A2 00A0 01A9 0020 C277
87D8:DDFF 4CA2 80A2 0506 B349
87E0:7526 76CA D0F9 A576 816A
87E8:1869 5085 7660 A900 221D
87F0:1869 14CA D0FB AAA0 3B75
87F8:00BD EF89 9900 01E8 20BA

8800:C8C0 14D0 F468 AA68 D36F
8808:A868 284C 0001 4898 DB9F
8810:488A 4820 B181 C90D FD15
8818:F02B 29DF C94D D032 1C31
```

Continued on page 66



Evaluating Educational Software

by Mark K. Sealey

How do you decide which software to buy for your pupils or students? It is likely that you will rely to some extent on reviews in magazines like BEEBUG. But when an interesting product which hasn't yet been reviewed comes your way, how do you decide? How do you interpret the review itself anyway? This month BEEBUG Education draws on the strongest points from the many schemes and checklists offered in recent years to provide a guide to choosing educational software.

BROAD CRITERIA

To start with, ask yourself what the program or package does. If this could be done better without the computer (for instance with pencil and paper), stop here and save your money. Although it is true that children are particularly motivated by the interactive nature of the computer, there is more and more evidence that they are also to some extent confused, by some screen conventions for example, and that this occurs on more occasions than we perhaps like to admit.

EQUIPMENT

Next, if you have decided to continue, check that you have and can use the equipment configuration required: most publishers make it very clear which disc format (40 or 80T, single or double-sided and so on) is required. But there are so many different Acorn computer systems that it is easy to overlook the fact that the software you have your eye on might not run on the Econet, or may require a second processor, or sideways RAM.

DOCUMENTATION

Many people would put the quality of any documentation high on their list of priorities when evaluating software. If there is no support when things go wrong or behave unpredictably, you are very unlikely to be able to make anything like the most out of what you have bought. Certain publishers (whose products tend to get reviewed more frequently in these pages) make a virtue out of clear, well-presented, attractive documentation; often this will have a teacher's ideas book and details of relevant resources to be used alongside the package. A technical appendix and booklists, as well as some mention of the Educational considerations underlying the software, are other good signs.

WHAT ARE THESE EDUCATIONAL CONSIDERATIONS?

This is not too difficult a question to answer. The most important thing is that the approach of the suite or program is child-centred, and likely both to appeal to the user and to work interactively. It is now accepted that we do not learn by listening, or even just watching, but by doing. The more the software involves the user the better.

To this end, it is helpful to know whether the software in question has actually been trialled by teachers and children. This fact certainly makes a difference to professional reviewers in their assessments. Is it clear that children have liked it and teachers found it easy to use? Does the overall personality of the package leave you with a positive feeling, one of having achieved something (assuming that you have the chance to try the suite out)? You would also be looking closely at the pace and grading of the program. Does it allow children to fit its use to their concentration spans? Are the sound, graphics and animation of a gimmicky type that will intrude after several sessions, or do they really add something to the presentation?

In certain cases it will be appropriate to ask if there are images, language or 'scenes' in the package which are offensive because of racist, sexist or class bias. Such images will almost always patronise the children and are to be avoided.

You may decide that the software will be too difficult or too easy for a particular age or ability range. This might be because of the language it uses, or it may lack a clear aim altogether. If not, is the purpose compatible with your own priorities in the learning situation? These may concern sharing, co-operation, ways of recording and so on.

Adventure games, for instance, often invite the users to work together in solving a particular problem. If you don't believe in this practice, the package may not suit.

It is helpful to examine just how the software fits into your existing curriculum. Does the package aim to introduce new material, test it, reinforce it, extend it or allow experimentation and/or investigation within it? Generally the latter is to be preferred because there will probably be greater scope for interaction.

It is also important that the program or programs leave the user in control. Again, research has shown that the better software does not lead pupils thoughtlessly along a predetermined path, but affords them choices and options to promote learning at their own pace.

Whatever the style and rationale behind the piece of software, it is vital that it enables the teacher to intervene and help the pupils, and that it is not so dense that, for example, time is wasted in learning a batch of unnecessary codes or spurious passwords. Does the suite permit adequate preparation and encourage time off the computer? Is it easy to link with other work under way and to plan follow-up activities?

EASE OF USE

It is imperative that the design and cueing systems (the prompts, error messages and input routines) are easy to follow. However good the original idea, there is always someone who will misunderstand an instruction, or forget for example that Escape interrupts the action. The better these pupils are catered for, the happier their experience of using the software is likely to be. In the cases where the overall layout (system of menus and sub-menus or icons) is simple and uncluttered, it is likely that thought has gone into planning the rest of the package. Try crashing it and see if the program can cope. How much is left after Break (and then Shift-Break or Ctrl-Break) have been pressed?

ROBUSTNESS

This aspect of a program is similar to the preceding. Bear in mind that it may not only be you, the specialist software user, who will want to work with the package. How well are errors trapped? How good is the feedback for 'unconventional' responses? Users who are not familiar with microcomputers must find the dialogue with the screen as helpful as any specialist's instructions. Another good sign is the way in which menus work. If an option is no longer relevant (because of an earlier choice, say, not to attach a printer), it might well be featured less prominently in the list, or better - disappear altogether. It should certainly be possible to return to the menu at any time to re-select and confirm choices.

There is a whole host of other similar points to watch. Is input permitted in both upper and lower case? If not, does the program turn off the Caps Lock? Does it disable Break? Does it, for instance, allow screen dumps (check that they work with your printer) to be interrupted by Escape or something similar? Check carefully how discs and files are organised. Ask yourself how easy it is to erase unwanted data, for instance, from within the program. Many non-specialist users will not want to learn the vaguaries of *DELETE <fsp> or *WIPE <fasp> and the rest. Is there provision for backing up data conveniently - perhaps also from within the suite? Is the product itself protected so that you cannot (legally or easily) make your own working copy?

FLEXIBILITY

It is arguable that the most successful educational software is that which allows teachers and students the greatest degree of choice in how they use it. For example, an adventure or simulation which has both its own scenario and allows you to create one of your own is preferable. So is one which permits saving of the game half way through, or where a secondary file can be opened, so that the pupil's 'moves' can be studied and/or diagnosed afterwards. These have more flexibility than software with a single unalterable route which must be taken each time.

CONCLUSIONS

Clearly not all the questions above will be appropriate to every piece of educational software. But these are the ones I have found the most helpful in evaluating software for pupils and students of all ages. We hope that you will write to BEEBUG Education with your own experiences and ideas too.

DISC SPOOLER UTILITY

Ever wished that your printer output could be redirected straight into a file on disc or tape? Derek Floyd has the answer with this compact and versatile machine code utility.

There are many programs which direct their output towards the printer or screen, but not to disc (or tape). In most situations this is all that is required. However, I have found several situations when I would have liked to divert the output from the printer, and send it to a disc file instead. The utility presented here will enable you to do just that.

DATABASES AND SPREADSHEETS

Databases and spreadsheets serve as a perfect example of this. There are always provisions to send output to a printer but they allow comparatively little control over the format of the output. The output can now be spooled to disc or tape, and can then be loaded back into a word processor or text editor, and altered or even incorporated into another document.

ALTERING AND RELOCATING MACHINE CODE

Another example can be found in the use of a disassembler. Altering and relocating a machine code program is simple enough providing that you have the source code (the original assembly language program). If you no longer have the source code, the machine code will need to be disassembled, typed in again, altered, and finally re-assembled. The majority of disassemblers allow disassembled code to be displayed on the screen or sent to the printer, but usually there is no provision to send the output to a file. Diverting the printer output to a file will save having to type in all the assembly language instructions again.

DIVERTING THE PRINTER OUTPUT

All the ingredients to redirect the printer output are available in the BBC's Operating System. Firstly, there is an FX call to direct the printer output in one of four directions.

```
*FX 5,0 Nowhere, a useful dump during
testing.
*FX 5,1 The parallel printer driver, the
default.
*FX 5,2 The serial printer driver.
*FX 5,3 A user defined printer driver.
*FX 5,4 Network printer.
```

There is also a user defined printer driver vector, or jump address, that can be used to point to the new driver, wherever we decide to put it. The following listing simply assembles a new machine code driver and saves it to disc. Once the driver has been loaded, entering *FX5,3 will divert the printer output to a specified file.

ENTERING THE PROGRAM

Enter the program paying particular attention to the assembler part, especially if you are not familiar with assembly language. Ensure that you save the program before you run it to avoid corruption in case of error.

USING THE PROGRAM

The program generates a short machine-code utility and automatically saves it to disc as 'DISCSPL'. The following information will then be displayed on the screen.

```
Disc Spooler saved as 'DISCSPL'
 It will operate from PAGE &900
 Syntax: DISCSPL <afsp>
 File can be closed by 'CLOSE # channel'
 Spooler is activated by '*FX 5,3'
 Spooler is de-activated by:
  '*FX 5,1' for parallel printers or
  '*FX 5,2' for serial printers.
```

To load the printer driver into memory and specify the file to which all output is to be diverted use the following star command:

```
*DISCSPL <filename>
```

The utility will generate an error if you do not

specify a filename. Make sure that the single space between the command and the filename is not omitted. If the file is successfully opened then this is announced together with the file channel (or handle) number in hexadecimal:

```
Spool file opened channel &11
```

Once a spool file has been opened, it may be activated with *FX5,3. From this point onward any output scheduled for the printer will be spooled to disc. The driver is de-activated by executing either *FX5,1 or *FX5,2 to restore your normal driver. Finally, the file should be closed with the command CLOSE#<channel>.

RECOVERING OUTPUT

The resultant file will be in a standard ASCII format and may be loaded into most popular word processors (e.g. Wordwise, View, etc.). If the file contains Basic or assembly language instructions then they will need line numbers. The easiest method of giving them line numbers is to load the file into a word processor and insert the word 'AUTO' before the first instruction. When you save the file from the word processor make sure that you spool the text out rather than just saving it (for View, save the file without rulers or embedded commands). If you then *EXEC the file from within Basic, line numbers will be provided automatically. Should you wish to do more elaborate things with the file it will be necessary to write your own program to read the file in, one line at a time, and deal with it accordingly. The following program demonstrates how to read in a file. In this example the file is simply sent to the printer but the program could be altered to direct the contents into another file, perhaps using a different format.

```
 10 REM SPOOLER
 20 REM D.R. Floyd
 30 REM May 1988
 40 :
100 CLS:VDU2
110 C%=OPENIN("filenme")
120 REPEAT
130 B%=BGET#C%:VDU1,B%
140 UNTIL EOF#C%
150 VDU3:CLOSE#C%
```

PROGRAM DESCRIPTION

The program is well structured so it should be easy to follow for those people familiar with assembly language. However, the listing is in such a style that errors should be easy to detect by anyone. The code can be segregated into three distinct blocks.

Lines 1160-1500. This routine initialises the spooler, setting the vectors to point to the new driver called 'spool'. The channel number is displayed using a short routine to avoid the different addresses in Basic I and II.

Lines 1510-1660. This is the new driver, which is not a printer driver at all, but simply sends characters to the disc buffer.

Lines 1670-1730. This is the display routine, which writes messages to the VDU screen. The message is pointed to by the zero-page address &80, and is displayed until either a &0D or &00 is found. The message is completed by a Carriage Return only if the terminator is &0D. The three messages are inserted into the assembled code using Basic indirection operators.

The last three Basic procedures can be considered as house keeping, and do not affect the utility itself. The final machine code utility just fits into one page of memory.

USING THE SPOOLER ON A CASSETTE SYSTEM

This utility works on tape as well as disc with a small number of alterations. Firstly, line 1090 should be changed to read &D00 so that the utility assembles and operates in an area not normally used by a cassette machine. Similarly, both occurrences of the number &900 in lines 1930 and 2020 should be changed to &D00.

Secondly, the utility may be run using */DISCSPL <filenme>. The command */ is read by the system as *RUN. This presents a problem, because the program looks for the space after DISCSPL to check that a filename follows. So, the numbers in lines 1180 and 1210

should be increased by one to &0709 and &070A to allow for the extra character in the command.

The only problems that may arise are when another program uses the same workspace, namely &70 to &82 and &900 to &9FF (&D00 to &DFF for tape). Now any printer output can be redirected to a file quickly and efficiently.

```
  10 REM Program   DISC SPOOLER
  20 REM Version   B1.57
  30 REM Author    D.R. FLOYD
  40 REM BEEBUG    May 1988
  50 REM Program   Subject to copyright
  60 :
 100 ON ERROR PROCerror: END
 110 MODE 7
 120 PROCtitle
 130 PROCinitialise
 140 PROCassemble
 150 PROCautosave
 160 PROCinstructions
 170 END
 180 :
1000 DEF PROCinitialise
1010 osfind%    = &FFCE
1020 osput%     = &FFD4
1030 osnl%      = &FFE7
1040 oswrch%    = &FFEE
1050 osbyte%    = &FFF4
1060 oscli%     = &FFF7
1070 filename% = &0070
1080 handle%    = &0082
1090 code%      = &0900
1100 ENDPROC
1110 :
1120 DEF PROCassemble
1130 FOR I%=0 TO 2 STEP 2
1140 P%=code%
1150 [ OPT I%
1160 .setup
1170 LDA #code% DIV 256
1180 STA &81:LDA &0708:CMP #&20
1190 BNE prompt:LDX #&0
1200 .floop
1210 LDA &0709,X:STA filename%,X
1220 INX:CMP #&0D:BNE floop
1230 .open
1240 LDX #filename% MOD 256
1250 LDY #filename% DIV 256
1260 LDA #&80:JSR osfind%
1270 STA handle%:BEQ cantopen
1280 .vectors
1290 LDA #spool MOD 256:STA &222
1300 LDA #spool DIV 256:STA &223
1310 LDA #message MOD 256:STA &80
1320 JSR display:LDA handle%
1330 LSR A:LSR A:LSR A:LSR A
1340 JSR print:LDA handle%:JSR print
1350 JSR osnl%
1360 RTS
1370 .print
1380 AND #&0F:ORA #&30:CMP #&3A
1390 BCC oscall
1400 .hex CLC:ADC #&07
1410 .oscall JSR oswrch%
1420 RTS
1430 .cantopen
1440 LDA #error MOD 256:STA &80
1450 JSR display
1460 RTS
1470 .prompt
1480 LDA #syntax MOD 256:STA &80
1490 JSR display
1500 RTS
1510 .spool
1520 CPY #&03:BEQ userprint
1530 RTS
1540 .userprint CMP #&00
1550 BEQ printok:CMP #&01
1560 JSR printok:CLC
1570 RTS
1580 .printok
1590 TXA:PHA:TYA:PHA:LDA #&91
1600 JSR osbyte%:BCS empty:TYA
1610 LDY handle%:JSR osput%
1620 .exit
1630 PLA:TAY:PLA:TAX:RTS
1640 .empty
1650 LDA #&7B:LDX #&03:JSR osbyte%
1660 JMP exit
1670 .display JSR osnl%:LDY #&0
1680 .loop
1690 LDA (&80),Y:JSR oswrch%:INY
1700 CMP #&0D:BEQ nldrts:CMP #&0
1710 BEQ drts:JMP loop
1720 .nldrts JSR osnl%
1730 .drts RTS
1740 .syntax
1750 ]
1760 $syntax="Syntax: DISCSPL <afsp>"+C
HR$13
1770 P%=P%+LEN($syntax)+1
1780 [ OPT I%
1790 .message
1800 ]
1810 $message="Spool file opened channe
l &"+CHR$0
```

```
 1820 P%=P%+LEN($message)+1
 1830 [ OPT I%
 1840 .error
 1850 ]
 1860 $error="Unable to open file"+CHR$1
3
 1870 P%=P%+LEN($error)+1
 1880 NEXT I%
 1890 ENDPROC
 1900 :
 1910 DEF PROCautosave
 1920 DIM save 30
 1930 $save="SAVE DISCSPL 900 "+STR$~P%+
" 900"
 1940 X%=save MOD 256
 1950 Y%=save DIV 256
 1960 CALL oscli%
 1970 *ACCESS DISCSPL L
 1980 ENDPROC
 1990 :
 2000 DEF PROCautosave
 2010 DIM save 30
 2020 $save="SAVE DISCSPL 900 "+STR$~P%+
" 900"
 2030 X%=save MOD 256
 2040 Y%=save DIV 256
 2050 CALL oscli%
 2060 *ACCESS DISCSPL L
 2070 ENDPROC
 2080 :
 2090 DEF PROCinstructions
 2100 CLS
 2110 PRINT"Disc Spooler saved as 'DISCS
PL'"
 2120 PRINT'"It will operate from PAGE &
900"
 2130 PRINT'$syntax
 2140 PRINT'"File can be closed by 'CLOS
E # channel'"
 2150 PRINT'"Spooler is activated by '*F
X 5,3'"
 2160 PRINT'"Spooler is de-activated by:
"
 2170 PRINT'"  '*FX 5,1' for parallel pr
inters  or"
 2180 PRINT'"  '*FX 5,2' for serial  pr
inters."
 2190 ENDPROC
 2200 :
 2210 DEF PROCtitle
 2220 CLS
 2230 FOR LA%=2 TO 3:PRINT TAB(13,LA%)CH
R$(141);"DISC SPOOLER":NEXT
 2240 PRINTTAB(9,7)"Printer output to Di
sc"
 2250 PRINTTAB(14,12)"written  by"
 2260 PRINTTAB(14,14)"D. R. Floyd"
 2270 PRINTTAB(7,21)"Press any key to as
semble"
 2280 G=GET
 2290 ENDPROC
 2300 :
 2310 DEF PROCerror
 2320 CLS:REPORT:PRINT" at line ";ERL
 2330 ENDPROC
```

BOXED IN THE CARPARK (continued from page 9)

```
 1800 IF W=A AND D1=0 AND D2=0 GOD=0
 1810 IF D=b AND L1=0 GOL=0
 1820 IF D=B AND L1=0 AND L2=0 GOL=0
 1830 IF D=b AND R1=0 GOR=0
 1840 IF D=B AND R1=0 AND R2=0 GOR=0
 1850 IF D1=3 AND W=A AND D=B GOD=0
 1860 ENDPROC
 1870 :
 1880 DEFPROCEND
 1890 PRINTTAB(2,29)"NOW PRESS SPACE BAR
 AND PUT IT BACK":VDU7:Solve=0:M$="J"
 1900 REPEAT UNTIL GET=32
 1910 PRINTTAB(2,29)SPC(36)
 1920 PROCdraw(0,M$,X%,Y%,W,D,x,y):Y%=Y%
+108:PROCdraw(2,M$,X%,Y%,W,D,x,y):PROCdr
aw(3,"",236,360,240,100,0,0):JX%=X%:JY%=
Y%
 1930 ENDPROC
 1940 :
 1950 DATA D,136,H,138,I,137,A,138,G,137
,C,139,A,136,G,138,I,136,I,136,B,139,B,1
37,D,139,D,139,G,137,C,137,C,138,I,138,D
,136,D,136,B,136,B,136,G,139,H,139,C,137
,C,137,A,137,A,137,I,138,D,138,D,137,E,1
38,E,138,J,136,G,139,G,139,H,136,C,139
 1960 DATA C,139,A,137,A,139,I,137,I,137
,E,138,B,136,D,139,E,137,B,138,D,136,E,1
39,I,136,I,136,A,138,C,138,A,136,C,138,H
,137,E,137,D,137,B,139,I,139,A,136,A,136
,C,136,C,136,H,138,E,138,D,137,D,137,B,1
37,B,137,I,139,A,139,C,136,E,136,B,138
 1970 DATA B,138,G,138,G,138,F,136,D,139
,H,139,B,137,G,138,F,138,D,139,D,136,H,1
39,H,139,B,139,B,139,G,137,F,138,F,138,B
,136,B,139,H,138,D,137,B,139,F,139,F,139
,E,137,A,137,A,138,I,138,J,138,B,136,B,1
36,D,136,D,136,F,139,H,139,E,139,G,139
 1980 DATA A,137,A,137,C,137,C,137,I,138
```

1st COURSE

Character Control (part 3)

Mike Williams concludes this mini-series on the use of Basic's string functions.

I hope you followed last month's article and the ideas on using string functions to justify both strings and numbers. Another application which uses very similar ideas is in file handling. Many filing systems use a sequence of fixed-length records each containing the same number of fixed length fields. Each piece of data (which is variable in length) needs to be either left or right justified (or padded out) within the appropriate field, depending on whether it is a string or a number. In fact, many file handling programs store all data in string format, converting to and from other formats when needed. The advantage of this approach is that the files themselves contain only a single data type and need just one set of procedures to handle them correctly. See our new series on file handling starting in this issue for more information on this subject.

CHARACTER SORTING

As a further example of the use of string functions I thought it would be instructive to look at a function for ordering the characters alphabetically within a single string. Thus, given a string of characters passed as a parameter, the function is required to return a string with the same characters ordered alphabetically. It is the ability of a function to return a value (or string) that makes that structure a better choice than a procedure in this case.

Although there are many sort techniques available, I propose to use one of the simplest and best known; the bubble sort. This technique 'bubbles' each item up to its correct position in the list. The function to do this may be written thus:

```
1000 DEF FNsort(string$)
1010 LOCAL n:n=LEN(string$)
1020 FOR i=n TO 1 STEP -1
1030 FOR j=1 TO i-1
1040 IF MID$(string$,j,1)>MID$(string$,j+1)
     THEN string$=LEFT$(string$,j-1)
     +MID$(string$,j+1,1)
     +MID$(string$,j,1)
     +RIGHT$(string$,n-j-1)
1050 NEXT:NEXT
1060 =string$
```

The bubble sort, as used in the function, compares adjacent characters in the string, and if necessary re-orders them so that 'lower-value' characters move towards the front of the list, all done with string functions alone.

The function depends upon the fact that characters may be compared together just as with numbers. In fact, it is their ASCII codes which are compared by Basic, and as we saw before, the ASCII codes representing the alphabet are in ascending numeric order. It also means that any digits will be ordered to appear before any genuine alphabetic characters, and that lower case characters will all appear after upper case ones.

As the program moves through the string, the character in each position j is compared with that in position j+1 and the two characters re-ordered if necessary. On each occasion that this happens, line 1040 ensures that the existing string is replaced by a new string consisting of that part of the original string to the left of the jth position, plus the character in position j+1, followed by the character previously in position j (thus reversing these two characters), and lastly the remainder of the original string to the right of position j+1.

All the work is done by the one line, and just the one string variable is needed for the purpose. The 'heaviest' character is moved to the right-hand end of the string, and the process repeated until all characters are in order.

If you want to try out this function, add the following lines:

```
100 MODE 7
110 INPUT"Enter any string: " chars$
120 PRINT FNsort(chars$)
130 END
```

This will allow you to enter any string (up to the maximum length of 255 characters allowed by Basic), and will then print the string with the characters in alphabetical order. If you add the following line to the procedure as well you will be able to see on the screen how the letters are gradually re-ordered:

```
1045 PRINTTAB(0,12)string$
```

There is one point to draw to your attention here, and that concerns the value of p (say) when used with string functions in the form:

```
LEFT$(string$,p)
```

In the sort function it is j and j+1 which are used. What happens if the value of p (or j or whatever) is out of step with the length of the string? If you try to extract more characters than exist all behaves as expected. For example, consider:

```
string$="abcdefgh"
PRINT LEFT$(string$,p)
```

IF p=20, for example, then "abcdefgh" will be displayed. The fact that the string actually contains fewer characters doesn't matter. If p=0 then again you get what you would expect, no characters at all. But what if p=-1 or any other negative number. You might expect that as with p=0 no characters at all would be displayed. In fact, the complete string would once again appear. Basic treats -1 here as a representation of the positive number 255 (256-1), as it assumes that the value of p in this context has to be positive.

Now you might say, at this stage, that using our example with p=-1 is hardly a sensible thing to do, and that's right, but when you are developing a program using string functions which include variables, this is a quite possible error that may inadvertently occur, and much confusion can arise if you are unaware of this possibility. In our sort example it may not be entirely clear what the full range of values is likely to be, or you might have used j-1 and j rather than j and j+1 to select characters. Understanding how Basic works can often help to resolve problems.

CHARACTER INPUT AND OUTPUT

If you want to enter simple character strings and print them out or display them on screen, then the standard Basic INPUT and PRINT will be quite sufficient. Once we start dealing with individual characters, other instructions generally prove more useful. GET and GET$ both input a single character, returning its ASCII code or the character itself. Experienced users generally find that GET is more useful than GET$.

Remember that when you use GET or GET$ nothing is echoed on the screen as happens with INPUT. This can sometimes prove to be quite useful. Alternatively, characters entered could be echoed with a different character altogether, a common practice for password entry, for example. The simplest way to echo just a single character on the screen is to use the VDU command - that's one of the reasons why GET is preferable to GET$. For example:

```
char=GET:VDU char
```

would input one character and echo it on the screen. If you put instead:

```
char=GET:VDU char+1
```

then any character input would be echoed by the one next in the alphabet, so 'A' would be echoed by 'B' and so on. Using GET and VDU, here is a simple password function.

```
1000 DEF FNcheckpassword(password$)
1010 LOCAL char,pos,string$
1020 pos=1:string$=""
1030 PRINT"Enter password:";
1040 REPEAT
1050 char=GET
1060 IF char=127 and pos>1 THEN
     VDU8,32,8:
     string$=LEFT$(string$,LEN(string$)-1)
     ELSE string$=string$+CHR$(char):
     pos=pos+1:VDU42
1070 UNTIL char=13:PRINT
1080 string$=LEFT$(string$,pos-2)
1090 =(string$=password$)
```

Calling this function with:

```
IF FNcheckpassword("BEEBUG") THEN . . .
```

would proceed only if the password prompted for were "BEEBUG".

The function is fairly simple and makes no other checks on the characters input or the

length of the string entered. It does, however,respond to the the Delete key (ASCII 127) by outputting the sequence:

<backspace><space><backspace>

That's the function of the VDU8,32,8. At the same time the last character is deleted from the input string. Characters that are accepted are echoed as asterisks (ASCII 42). The function bears quite strong similarities to the input function I gave in the first article in this series (Vol.6 No.9).

GET and VDU work together quite neatly. Do remember, though, that VDU (unlike PRINT) will not automatically output a Carriage Return/Linefeed sequence at the end of a string of characters. You will need to provide this explicitly (VDU13,10).

Another bonus that results from using ASCII codes with GET, and more particularly with VDU, is that it becomes quite simple to deal with non-printing characters, or indeed any character which cannot be generated from the keyboard. Such characters are the ones with codes from 0 (the so-called null character) up to 31. These are the characters that are entered as Control codes from the keyboard, like Ctrl-N or Ctrl-B for example. They can be readily output by a program using VDU14 or VDU2 respectively. All these codes are listed in the User Guide under the heading of VDU codes. A good many of them duplicate statements in Basic like MOVE and DRAW. All in all, both GET and VDU can prove most useful and effective for many purposes as your programming skills develop.

One further single character input function is INKEY. This is like the GET function (and likewise INKEY$ is similar to GET$), but it waits for input for a limited time only. If no character is entered within the time specified, then a -1 is returned. GET will wait for ever.

THE EVAL FUNCTION

Finally, I would like to draw your attention to the EVAL function in BBC Basic. This is an often misunderstood and little used facility, and yet it can be extraordinarily powerful. It is also quite unlike any of the other functions which I have covered, and really deserves a whole article to itself (see First Course Vol.4 Nos8 & 9 for a much more detailed discussion of this).

EVAL accepts a single string argument and attempts to evaluate this as an expression. Thus if we write:

```
formula$="u*t+0.5*a*t^2"
s=EVAL(formula$)
PRINT s
```

then the formula specified as a string will be evaluated by the EVAL function using the current values (in this case) of u, t and a.

One easy to understand application of this principle is the writing of a program to draw a graph of a function input by the user, for example:

```
100 MODE 0
110 VDU5,29,640;512;
120 INPUT"Give function of x: " f$
130 FOR x=-6.4 TO 6.4 STEP 0.1
140 MOVE 100*x,100*EVAL(f$)
150 PRINT "*";
160 NEXT x
170 END
```

This is indeed quite crude, but if you run this short program and enter any reasonable function of x - for example, try `3*SIN(x)+SIN(3*x)` - you should get some kind of result. Without EVAL there would simply be no way of entering a formula from the keyboard for evaluation within a program.

The graph of the function specified will be plotted with asterisks - you could change this. The VDU codes at line 100 select text printing at the graphics cursor and move the graphics origin to the centre of the screen. A slightly more elaborate version of this program is included on the magazine disc/tape. The EVAL function is capable of very much more that I have covered here, and really is worth investigating further to exploit its full power.

This concludes our present discussion of string functions. Do let me know if you have any questions arising out of these three articles, or if there are any other topics you would like to see covered in future First Course series. B

A full implementation producing the most compact code in the fastest compilation time, for the BBC Micro and Master series.

"...the Beebug implementation is superior in so many respects..."
"The Beebug C system is far superior to that provided by Acorn"
"...the Beebug version has to be the best buy"
A&B COMPUTING FEB 88

"...Beebug C is about five times faster at compilation and linking than the Acornsoft version and produces code at a fraction of the length."
MICRO USER NOV 87

The Language C is now available from Beebug for all users of the BBC Micro and Master. Beebug C conforms to, and extends beyond the Kernighan and Ritchie standard, producing fast, compact code and supporting full floating point maths. A comprehensive set of library functions are supplied on disc, conforming to the proposed ISO standard.

Features Include:

- Runs on a standard 32K BBC model B, B+ or Master 128
- 40/80Track DFS and ADFS compatible
- Support for Acorn operating system (vdu, osbyte, mode etc.)
- Powerful command line interpreter with over 20 commands & qualifiers
- Expandable run-time library on disc containing nearly 100 functions & macros
- Full macro-handling facilities
- Debugging facilities and helpful error messages

Technical Summary

Beebug C is a full implementation of the Kernighan & Ritchie standard. The following is a summary of the full specification.

Expressions:	*, &, -, !, ~, ++, --, sizeof, ->, *, /, %, +, -, >>, <<, <, >, <=, >=, &, ^, |, &&, ||, ?:
Assignments:	=, +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=
Declarations:	char (8 bits), int (16 bits), short (8 bits), long (32 bits), float (32 bits), double (32 bits), unsigned, void, pointer, auto, static, extern, typedef
Statements:	if, while, do, for, switch, case, default, break, continue, return, goto, struct, union
Preprocessors:	#define, #undef, #redef, #include, #if, #ifdef, #ifndef, #else, #endif, #line, #pragma
Library:	nearly 100 library functions, plus a full range of header files - h.stdio, h.stdlib, h.string, h.ctype etc.

BEEBUG C is supplied on two 16K ROMs & a disc. ***(Specify 40/80T)***

Members Price
£44.25

Non-Members Price
£59.00

To write C programs you will need a text editor or word-processor such as View, Wordwise, InterWord etc. Beebug C is supplied with a detailed user guide, however this does not teach C, and a basic knowledge of the C programming language is assumed. The definitive book on C is **The C Programming Language** by Kernighan & Ritchie available from BEEBUG.

Personal Ads

Master 128 £290, Morley EPROM Programmer £22, Clares Artroom 80 track ADFS. Tel. (04243) 4500.

6502 Acorn Processor and DNFS and HiBasic £120. Microtec colour monitor £135. Aries B20 and B12 and 16K RAM £65. Zenith Green Screen Monitor £45. Viglen cartridge system and 8 cartridges for Master £12. Romit ROM £15. Acorn/Mirle complete business system (7 Discs) £42. Joystick £5. Watford Mk II ROM board £15. All internal units can be fitted free if required. Tel. (040 381) 4976.

Epson RX-80 F/T dot matrix printer complete with leads, instructions and original box. Good condition, very well made and reliable. £90 ono. Tel. 01-341 2187.

WANTED: 1986 Issues of Micro User. Tel. (0788) 822508.

Cumana 40T drives - pair c/w PSU, 50 x 40T SSDD discs - K125. Tel. (09323) 42991.

Master 128 Turbo, twin double sided drives, replay £550. Microvitec 1451 monitor £190. Acorn teletext (ATS) £75. ROM boards, ROMs, books, software, too much to list. Tel. (0543) 254805 (eves).

Miracle Technology Modem WS2000 with DATABEEB communications ROM and RS 423 lead £40. Tel. (0473) 213907 (after 6 pm).

Master 65C102 6502 "Turbo" Upgrade, plus Tubelink "Advanced Basic" ROM and Hibasic87 ROM also Tubelink utility disc all in original packages with all manuals. £100. Tel. (0322) 64761 (eves only).

BBC B Issue 7, DFS, ADFS, 3.5" Mitsubishi 80T drive with PSU, Replay tape/disc ROM, Microvitec 1451AP-DS med. res colour monitor with TV Tuner, many games, utilities, FORTH, etc, User and Advanced Guides £450. Tel. (0327) 704401.

BBC B Series 7, Solidisc DDFS, Microvitec Cub Monitor, Wordwise+, Paul Beverly's Continuous Processing ROM, tape recorder together with all manuals, etc. £475 ono. Tel. (0502) 2277.

D.R. DOS plus manual £15. Mastering DOS Plus £5. AMX Super Art (Master) with mouse £20. DDCPM £10. Acorn Prestel adaptor £35. Wordstar Professional for Acorn 280 £50. Exmon II £10. BEEBUG Vol 1 - Vol 6 £15. Masterfile II £8. Elite £5. Creative Sound £8. Microtext £25. Acheton £5. Tel. (0533) 312661.

Electron/BBC games, originals, mostly cassette, £1-£4. SAE for list. Danny Langton, 13 Whitmore Close, New Southgate, London, N11 1PB.

PMS multi-font NTQ. Two ROM set, utilities disc and User Guide. For use with and Epson MX80 printer to give Near Text Quality printing £25. Tel. (0474) 363503.

BBC B Issue 7, 1.2 OS £195 ono. Tel. (0724) 720675.

Solidisk SWRAM 32K and manual. Watford Electronics double density filing system board and ROM and manual, also WE DFS ROM. OPUS 40T S/S disc drive and manual and utilities disc. Cassette software. The Advanced User Guide for BBC. Z80 2nd processor and bundled software. Wordwise ROM and manual. Shinwa CP 80-TI printer, 80 cps, Epson compatible, NLQ mode. AMX mouse with Super Art, manual, discs, etc (for Master 128). Offers invited. Tel. (0229) 62566.

Acorn Electron with Turbo driver, Plus 1, Plus 3, ACP Advanced Sideways RAM, ACP DFS E00, ACP Plus 5 and AMX Mouse, Beebug Toolkit ROM, View, ViewSheet, Database, Elite, all manuals and books, 25, 3.5" discs with various software. Will split. £300 ono. Tel. (0480) 61668 (after 6 pm).

32K ROM/RAM board £30. 32K shadow RAM £25. AMX Mouse and Super Art £40. Advanced Control Panel £12. Masterfile II £10. Original cassettes and discs £40. Thirty DS/DD discs (blank) £25. Cassette recorder and Joystick £15. High res monitor £60. Complete Beebug in covers £28. Tel. (044282) 4600.

Opus single sided, 40T disc drive without power supply £65. Tel. (041 942) 7197.

Enigma chip £20 without Viglen cartridge or £23 with cartridge. Chip only works on BBC or BBC+. All relevant manuals. All offers welcome. Tel. (0904) 707447.

WANTED: 2 disc drives DS DD, preferably not Cumana, unboxed without power supply. Willing to pay £50 each. Tel. (0273) 723467 (eves).

Aries B32 complete with ROM chips and 6502 CPU £65. Aries B12 with B12c (for use without B32) £25. Viglen BBC B keyboard case, with 2 metre coiled connector £10. All inclusive postage and manuals. Tel. (0932) 226076.

AMX Mouse/Super Art £30. BCPL + SAG £6. Dumpout 3 £3. Transfer ROM and cassette including aviator, etc, £5. Tel. (0502) 82564.

288 with mains adaptor and 128K RAM and 128K EPROM. New, still in boxes £325. Tel. (0782) 316763 (eves).

Interword with manual £35. Tel. 01-803 6763 (after 6 pm).

Continued on page 36

REALTIME SOLIDS MODELLER

"Realtime Solids takes all the hard work out of truly impressive graphics displays on your BBC micro. It doesn't take much imagination or effort to create extremely complex shapes from very simple beginnings....full hidden line removal and hardly a trace of flicker"
- BEEBUG (APRIL '88).

The complete 3D Solids/Wireframe package for architectural design, interior design, engineering design, teaching CDT and 3D geometry, molecular modelling, mathematical plots, scientific processing and high speed flicker-free 3D animation.

Hidden Surface Removal can be performed for full colour realistic solid displays at high speed. The 3D solid images can then be incorporated into other art designs.

Supports all plotters including HP-GL, Plotmate, Penman II, Hewlett Packard, Epson HI-80, Hitachi 672, Watanabe, Graphtec, Calcomp, Seckonic, Houston DMP, Roland and paper sizes from A0 to A4 for professional results.

The package consists of a 32k Realtime Graphics Language rom, Solids Design disc, Wireframe Design disc, Multi-plotter and printer driver disc, Demonstration disc, Applications disc, Database library disc and a fully comprehensive 150 page manual designed for complete beginners and experts alike.

Design facilities include 3D Solid/Wireframe Editors to design objects with lines and surfaces, create symmetrical objects by Sweeping or Extruding simple sections, use objects as building blocks to create more complex objects which in turn may be used as building blocks, specify individual surface and line colours for multi-coloured objects, dynamic 3D viewing in any graphics screen mode and a Data convertor for interfacing to other CAD systems and applications.

The Solids Realtime Graphics Language (RGL) rom provides 52 star commands to write your own 3D applications. The facilities include 3D Rotation, Scaling, Translation, Perspective and Isometric displays, Orbiting, 3D Turtlegraphics, Geometric processing, unique 35,000 pixels/sec line generator (faster than Acorn's 9000/sec) for high speed 3D animation and many more options.

The Solids RGL is compatible with all the BBC graphics screen, plotting and colour modes including Shadow screen and is up to 5 times faster than the original RGL. The 32k RGL rom board can be plugged into any 16k rom socket.

Minimum requirements: BBC B/keyboard/single drive. Also compatible with BBC B+, Master series, all versions of DFS & DDFS. The system will also take advantage of a 6502 2nd Processor/Turbo or AMX/QUEST Mouse.
REALTIME SOLIDS MODELLER: £89.95.

3D CAD/ANIMATION SYSTEM

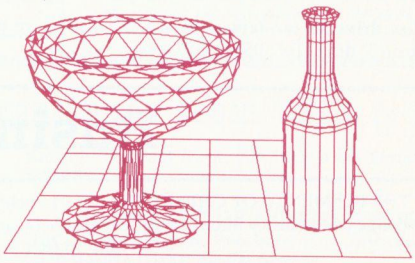
ARCHIMEDES & BBC VERSIONS

Once again Silicon Vision steals the lead with this incredible offer. It's true we're offering our international, best-selling wireframe 3D Graphics Development System (3D GDS) at an incredibly low price. No we haven't gone mad, it's simply our way of introducing you to our unbeatable 3D Solids Design & Animation Systems.

3D GDS is a full blown 3D CAD & Animation system that can handle wireframe models of any complexity. The package consists of a Design & Animation disc, a database disc to get you started, and a comprehensive 95 page manual. The standalone animation facilities for the BBC include a unique 30,000 pixels/sec line generator for high speed flicker-free 3D animation - ideal for games & simulations.

In the future, you may find that you need a 3D Solid Modelling system for more professional results or more powerful animation facilities for better effects.

Well, then you'll find no better company to do business with than Silicon Vision. As Europe's fastest growing CAD software house, there's no-one more capable of satisfying your future needs. In the meantime we offer you 3D GDS with our compliments. Worth over £25 it can now be yours for only £12.95 (BBC B/ B+/ Master) or £19.95 (Archimedes 305 to 440). Super-Plot plotter driver: £15.95 (BBC B/ B+/ Master).



SUPER-DUMP

1920x1024 Resolution Breakthrough! Near Plotter Quality hardcopy at a fraction of the cost.

The ultimate printer driver which provides 1920x1024 resolution for high quality printout as opposed to the 640x256 resolution limitation of most printer dumps.

Fully compatible with the Realtime Solids Modeller, 3D GDS and all other CAD packages and applications that can produce a VDU text file of a screen image. Images can also be scaled, positioned and previewed before printing.

SUPER-DUMP: £15.95 (BBC B/B+/Master). Enquire for Archimedes version.

SPECIAL OFFER TO BEEBUG MEMBERS ONLY (Quote Membership Number when Ordering).

3D CAD/ANIMATION (BBC): £11.95.
3D CAD/ANIMATION (Archimedes): £17.95.
REALTIME SOLIDS MODELLER (BBC): £75.
SUPER-DUMP (BBC): £14.95.
SUPER-PLOT (BBC): £14.95.

SILICON VISION

All prices are fully inclusive.
Contact your local dealer or order directly by cheque/P.O./Access/Mastercard/Eurocard from:
SILICON VISION LTD, SIGNAL HOUSE, LYON RD, HARROW, MIDDX, HA1 2AG.
Tel: 01-422 2274 or 01-861 2173. Fax: 01-427 5169. Telex: 918266 SIGNAL G.

Personal Ads (Continued from page 34)

BBC B+ 128K (64K S/RAM), includes 1770 DFS, ADFS, View, GXR, Viglen Dual DS 40/80 drives, books, inc 30 hour Basic, Joystick. Data recorder and all leads. Dust cover. Many original software titles, eg Elite, Frak, printer driver generator, Monopoly, Connect Four, etc. Too much to mention. Over £2000 worth of software. Tel. (0707) 872005 with any reasonable offers.

Producer/Engineer with own studio and instruments requires musician/songwriter for talented female vocalist. Computer Concepts Wordwise Plus ROM and manuals in perfect condition. Sensible offers please! MTR642 mixer, TR606 drum machine, ACES 32-way patch bay, CUTEC 8Urack mount. Very good condition. £300 ovno. Tel (0494) 716694 (after 6 pm).

Printer Smith Corona, fast teletext 80 £100 or ono and BBC Model B with data recorder - with 14 games, Elite, etc and Joystick £180 ono. BBC manuals £1. Tel. (0753) 75141.

BBC B issue 7, DFS, Cumuna 100K drive, Wordwise, Toolkit, Communicator ROMs plus software, games, books £290 or offers for individual items. Tel. 01-699 5087.

3" disc drive in tandem with a 5.25" disc drive and nineteen 3" discs. Tel. (0606) 3589 (eves).

Disc drives: dual DS40/80 and psu £160, single SS40 £40, Solidisk 256K 4 Meg £60, DFDC and ADFS £40, Viglen BBC console £35, Speech upgrade £20, View 3.0 £40,Viewsheet £25, GXR (B) £20, Wordwise £15, BBC psu £40, Drive psu £25, Prism 1000 modem and Watford Modem 84 ROM £40. Tel. 01-903 5881.

Printer Microline 80, dot matrix, tractor and friction drive £100. Interword, boxed with instructions £35. Solidisk 128K sideways RAM with disc £25. Tel. (0672) 810625 (eves).

BBC/Master software. ACP £20. Comal £24. Master cartridges 1/2 price. Numerous books. Voltmace Joysticks max £10. Basic Editor £15. Many others, all original. Tel. (0295) 65262.

Viewsheet, Viewstore, System Delta Card Index, Mailshot and Reporter, Gemini Office Master, all with manuals and also the System Delta programmers reference book. Acorn Prestel Adaptor (no software), Cumana 40 track 100K discdrive with PSU. Other books. All open to offer. Tel. (0707) 50568.

Kaga high resolution green monitor, Watford 32K ROM/RAM board, Joysticks, Sweetalker, complete bound Beebug, games, books and magazines £150 or separate. Tel. (044 282) 4600.

Business Ads

THE ELEMENTS OF CHEMISTRY (C) This 'chemical blockbusters' using the periodic table will test your knowledge of chemistry interactively, exhaustively, and enjoyably. A game for two players: supplied on disk for the BBC Master Series (specify 5" or 3.5").
*includes over 300 m/c and single answer questions.
*first class revision aid for *GCSE/A level.
Price £15.00. Order from: ***New-Life Software, 7 Fulmar Close, Bradwell, Gt Yarmouth, NORFOLK, NR31 8JG.***

MESSAGE MASTER for bulletin board users. Help reduce lengthy phone calls by preparing messages off-line and then transmitting them error free to the bulletin board(s) of your choice. Messages can be written with a wordprocessor like View or Wordwise, or using the editor provided. Once a message has been written it can be saved away onto disk for further editing or transmission. Only £8!

SCREEN COMPRESSOR. Compress/Decompress graphics screens within a second using this program. Ideal for title pages and demonstration programs. Complete with instructions for use within your own programs. Only £5!
Send SAE for details or cheque/PO to
TurboSoft, 9 The Headland, East Goscote, Leicester, LE7 8QT.

BEEB-PLANNER, Version 8, CPA program, Time Analyse 250 Activities, calculates project cost, three calendars, various reports, uses sideways RAM, £39.95. ***E J Sheffield, 8 Langdon Close, Camberely, Surrey, GU15 1AQ.***
Send SAE for details.

TRAINING COURSES

A New Service From BEEBUG

We are currently planning a range of one day courses to be offered to BEEBUG and RISC User members. These courses can be tailored to suit individual needs, as well as the needs of groups from business or education. Courses can be held at St Albans, your own home or your business premises, and charges will vary accordingly. Courses will be aimed at three levels to suit everyone from the complete beginner to the professional who wishes to gain experience quickly in a particular area.

If you would be likely to take advantage of any such courses, it would help us considerably if you would indicate (using the form below, or by letter or postcard) those areas of interest, and the level. We can only offer those courses for which there is sufficient demand.

	NOVICE	INTERMEDIATE	EXPERT
The Archimedes			
The Master/Compact			
Upgrading			
Peripherals			
Econet			
The 'C' Language			
Basic Language			
Assembly Language			
Wordprocessors			
Spreadsheets			
Databases			
Other (Please specify)			

Preferred venue:

Your Home ❑

Your Business Premises ❑

St Albans ❑

Name.....

Address.....

.....

.....

Completing this form does not commit you to attending any courses. The information is for statistical purposes only. You will be sent more specific details later.

BBC USER GROUP INDEX (continued from Vol.6 No.10)

WEST GERMANY

GERACUS - German Acorn User Club.
Contact Roul Sebastian John, Wasserstrasse 475, 4630 Bochum 1, West Germany.

HONG KONG

'BBC Micro Computer Users Informal Liaison Group.' Contacts: R Lumb (5-921985) P Monger (3-7217585)

Acorn Computer Users Society of Hong Kong
Meet on the first Wednesday of each month at the Brainchild Computer Centre, Far East finance centre at 7.30 p.m. Contact the society at P.O.Box 13330, Central Post Office, Hong Kong.

NEW ZEALAND

BBC-Acorn Computer User Group of NZ.
PO Box 9592, Wellington, New Zealand.

B.M.Wilkinson, Einstein Scientific, 177 Willis Street, PO Box 27138, Wellington, New Zealand. Tel: 851-055.

NORWAY

Oivind Grennes, **BBC Norway**, O-INFORM, Postboks 716, 3191 HORTEN, Norway.

PAKISTAN

Anyone interested in forming a BBC User group in Karachi Contact Capt. Z.A.Kidvai on Karachi 540986.

REPUBLIC OF SOUTH AFRICA

BBC User Group of Pretoria. P.O. Box 32798, Glenstantia 0100, South Africa

Pretoria BBC User Group, Contact: Stan Miller, P.O.Box 117, Montana, 0151 Pretoria, Rep.of S.Africa.

The Durban BBC User Group, P.O.Box 148, Umhlanga Rocks, 4320, South Africa. All enquiries to the secretary, Frank Calboutin.

Tygerberg BBC User Group (Tygerberg) For Electron, BBC and Master. R.P.Donovan (Secretary) 16 Bakker Street, Welgemoed, Bellville, 7530 South Africa. Tel: 021-953 2210

ZAMBIA

BBC User Group. Contact J.Maurice Brown. For enquiries in or near Zambia: c/o Britsh High Commission, P.O. Box 50050, Lusaka. FOR enquiries from UK: c/o F.C.O. (Lusaka), King Charles Street, London. SW1A 2AH.

ZIMBABWE

Green Screen Club
P.O.Box U.A.393, Union Avenue, Harare, Zimbabwe

RISC USER

The Archimedes Support Group

Our new Risc User magazine has now had six successful issues and is enjoying the largest circulation of any magazine devoted to the Archimedes. The list of members seeking support from the Risc User group is growing rapidly and at present we believe that it includes over half of the Archimedes owners.

Existing Beebug members, interested in the new range of Acorn micros, may either transfer their membership to the new magazine or extend their subscription to include both magazines. A joint subscription will enable you to keep completely up-to-date with all innovations and the latest information from Acorn and other suppliers on the complete range of BBC micros. RISC User has a massive amount to offer, particularly at this time while documentation on the Archimedes is still limited.

Here are some of the topics covered in the first six issues of RISC User:

INTELLIGENT AUTO-CONFIGURE
A program which reconfigures your machine to suit any particular application.

A series on **BASIC V**

OFF-THE-SHELF
A look at some of the products available now for the Archimedes

BBC TO ARM SOFTWARE CONVERSION

ARCHIMEDES VISUALS
The Mandelbrot Set
A Fractal Pattern Generator
And more short Basic routines producing stunning visual effects.

A WINDOW ON THE ARCHIMEDES
An introduction to the Archimedes WIMP environment for Basic programmers.

BEEB TO ARCHIMEDES LINK

3D GRAPHICS
A series of basic programs creating and manipulating 3D objects.

INTRODUCING ARM ASSEMBLER
A series dedicated to ARM machine code.

SOUND AND MUSIC
How to use the Archimedes sound system.

LOGISTIX FOR THE ARCHIMEDES
A major spreadsheet reviewed.

RISC USER MEMORY EDITOR
A utility for editing user RAM

GETTING THE ARCHIMEDES ON-LINE
A survey of comms packages.

ARCHIMEDES INTO PC
A look at the PC emulator.

Don't delay - Phone your instructions now on (0727) 40303

As a member of BEEBUG you may extend your subscription to include RISC User for only £8.50 (overseas see below).

Name:.....
Memb No:.....
Address: .....
.....
.....
.....

SUBSCRIPTION DETAILS

Destination	Additional Cost
UK,BFPO &Ch Is	£ 8.50
Rest of Europe and Eire	£13.00
Middle East	£15.00
Americas and Africa	£17.00
Elsewhere	£19.00

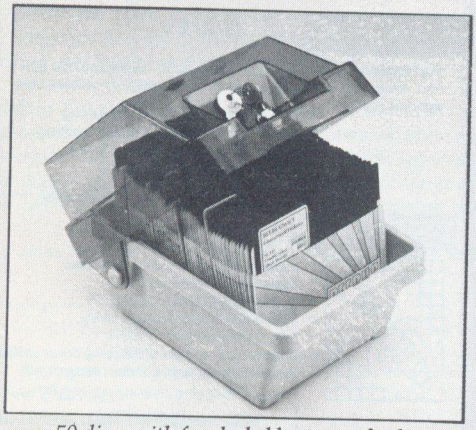
*I wish to receive both BEEBUG and **RISC User**.* *I enclose a cheque for £ ..... or alternatively*

I authorise you to debit myACCESS/Visa/Connect account: _____/ _____/ _____/ _____/

Signed: ..... *Expiry Date:* _____/ _____/

Send to: RISC User, Dolphin Place, Holywell Hill, St Albans, Herts AL1 1EX, or telephone (0727) 40303

BEEBUG Discs - The Ultimate in Quality & Reliability



50 discs with free lockable storage bock

50 Single Sided 40T Double Density Discs — **Only £35.60**

50 Double Sided 80T Quad Density Discs — **Only £39.90**

Prices shown are members prices and include VAT.

BEEBUG discs are manufactured to the highest specifications and are fully guaranteed.

Our Guarantee

We confidently offer a lifetime data guarantee and will replace any disc with which you encounter problems. We have found that the standard of quality control at the factory makes this necessity very rare.

40 Track Single Sided Double Density

	Price	Members Price	Order Code
10	£9.37	£8.90	0657
25	£23.00	£21.85	0661
50	£37.50	£35.60	0665

80 Track Double Sided Quad Density

	Price	Members Price	Order Code
10	£10.42	£9.90	0660
25	£26.20	£24.90	0664
50	£42.00	£39.90	0668

Please send me _____ (qty) _____ (stock code) at £_____ (unit price)
UK post 10 £1, 25/50 £3.75. Overseas send same price inc. UK post & VAT
I enclose a cheque for £_____ /Please debit my Access/Visa card £_____

Expiry date: ____

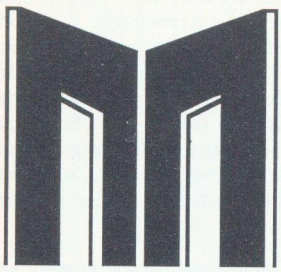
Name _____ Memb No. _____

Address _____

_____

BEEBUG
Dolphin Place,
Holywell Hill,
St. Albans,
Herts.
AL1 1EX

☎ (0727) 40303



THE MASTER PAGES

Devoted to the Master Series Computers

This month's Master pages provide a detailed examination of the use of extended vectors, the method by which programs may call routines residing in sideways RAM or ROM. Claus Alsted explains all. In addition we have included a review of the long awaited Advanced Reference Manual for the Master, published not by Acorn as expected but by Watford Electronics. Finally we have included some further hints and tips specially for Master and Compact users.

We would still welcome more contributions for publication in future Master pages.

MASTER SERIES

VECTORING AROUND

In this article, Claus Alsted, explains how to call routines stored in sideways RAM, and presents a utility to assist the process.

Many people are put off using sideways RAM because of the problems of accessing it, but with a simple trick all these are resolved. In BEEBUG Vol.6 No.1, Bernard Hill showed how to use extended vectors when writing programs to run from sideways RAM. This article goes one stage further and shows how any routine in sideways RAM can be called via an extended vector. This will allow, for example, a lengthy machine code program to be loaded into sideways RAM and then called from Basic.

The key to the technique presented here is the way in which the MOS calls routines in sideways RAM (or ROM), and we shall look at this stage by stage. This is demonstrated in a short example program, and a further utility is included to assist in the use of this technique in your own programs.

HOW THE MOS CALLS SIDEWAYS SOFTWARE

Firstly, when an operating system routine is called, either overtly, such as with JSR &FFEE (OSWRCH) to print a character, or covertly, as when an event is generated, the MOS reads the address from a vector in page 2 of memory, and jumps to that location. For example, on a standard Master 128, the vector for OSWRCH is at locations &20E and &20F. The contents of these two locations is &E822, which is the address in the MOS of the write character routine. Finally the MOS jumps to this routine.

This method of indirecting through a vector will not work if the routine to be called is in the sideways ROM/RAM area, because in such cases the appropriate ROM must be paged in first. To overcome this the MOS uses a special calling technique called double indirection. Firstly, the vector in page 2 is changed to point to a routine in the MOS ROM between &FF00 and &FF4E. This routine then reads the address to be called, and its ROM number, from a 3 byte 'Extended Vector' in page &D. Finally the MOS pages in the ROM and calls the routine.

There are 27 vectors in all, numbered from 0 to 26, the vector number being used to determine the address of the page 2 vector, the address of the &FF00 routine, and the address of the page &D extended vector. For vector number 'X', its page 2 vector is held in the two bytes starting at &200+2*X, the extending routine is at &FF00+3*X, and finally the extended vector is held in the three bytes from &D9F+3*X onwards. For example, the OSWRCH vector, which is called WRCHV, is vector number 7, which means that the main vector is at &20E

and &20F, the extending routine is at &FF15, and the extended vector at &DB4 - &DB6.

The best way of calling your own machine code routine residing in sideways RAM is to set one of the unused extended vectors in page &D to point to the routine, and then call the routine by calling the appropriate address in page &FF. For example, if the address of a machine code routine in sideways RAM was stored in the extended vector at locations &DDB - &DDD, which is the keyboard vector, then the routine could be called from Basic by CALL &FF3C. An example of this is given in listing 1, which assembles a short piece of code to print a message, copies it to sideways RAM, and then calls the routine through an extended vector.

CHOOSING EXTENDED VECTORS

The problem with using this technique is in choosing which extended vector to use. If you use a vector that has already been extended by a sideways ROM for its own use, then as soon as that vector is changed, the computer will crash. Which vectors are used by ROMs inside the computer depends very much on the ROMs installed, although on a Master the filing system vectors will always be extended. Listing 2 is a machine code utility that will list out the contents of each vector, so that you can see if it is extended. When run, the program saves a machine code program to disc under the name 'VECTOR'. Once the code is assembled, typing *VECTOR will list all the vectors along with their contents. Alternatively, the command can be followed by a vector name or number to list information on one particular vector. For example, *VECTOR CNP or *VECTOR 23, both of which produce the same results.

The display produced by *VECTOR consists of eight items for each vector. The first two are the vector number, mentioned earlier, and the vector name. The next two are the address of the vector in page 2, and its current contents. Then comes the address of the extended routine in page &FF, and the address of the extended vector in page &D. Finally, the display shows the current value of the extended vector, including the ROM number to which it points.

The best way of telling if a vector is extended or not is to look at the value of its main vector in page 2. If this is between &FF00 and &FF4E then the vector is being used in extended mode, and you shouldn't use it yourself. Any other value means that the operating system isn't using that vector in its extended form, meaning that it can be used freely by your own program. The only exception to this is the FSC vector. Because of the way the operating systems handles temporary filing systems, this vector appears not to be extended, while in fact it always is.

By using *VECTOR to find out the current state of the vectors, it should be very easy to choose which extended vector is best used to call a routine in sideways ROM or RAM.

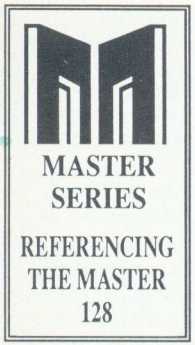
Listing 1

```
  10 REM Program Extended Vector Demo
  20 REM Version B1.0
  30 REM Author  Claus Alsted
  40 REM BEEBUG  May 1988
  50 REM Program subject to copyright
  60 :
 100 PROCassemble
 110 :
 120 REM Copy code to SRAM
 130 *SRWRITE 900 A00 9234 5
 140 :
 150 REM Setup extended vector
 160 ?&DDB=&9234 MOD &100
 170 ?&DDC=&9234 DIV &100
 180 ?&DDD=5 : REM Sram bank 5
 190 :
 200 REM Call routine
 210 PRINT'
 220 CALL &FF3C
 230 END
 240 :
1000 DEFPROCassemble
1010 osasci = &FFE3
1020 FOR pass%=4 TO 7 STEP 3
1030 P%=&9234 : O%=&900
1040 [OPT pass%
1050 LDX #0
1060 .loop
1070 LDA text,X:BEQ done
1080 JSR osasci:INX:JMP loop
1090 .done
1100 RTS
1110 :
1120 .text
1130 EQUS "Hello World": EQUB 13
1140 EQUB 0
1150 ]
1160 NEXT
1170 ENDPROC
```

Listing 2

```
   10 REM Program Vector Lister
   20 REM Version B1.0
   30 REM Author  Claus Alsted
   40 REM BEEBUG  May 1988
   50 REM Program subject to copyright
   60 :
  100 ON ERROR GOTO180
  110 DIM code 1000
  120 PROCassemble
  130 PRINT'"Press Space to save machine
 code"
  140 REPEAT UNTIL GET=32
  150 OSCLI ("SAVE VECTOR "+STR$~code+"
"+STR$~O%+" 410 410")
  160 END
  170 :
  180 ON ERROR OFF: IF ERR<>17 REPORT:PR
INT" at line ";ERL
  190 END
  200 :
 1000 DEF PROCassemble
 1010 gsinit=&FFC2:gsread=&FFC5
 1020 osargs=&FFDA:osasci=&FFE3
 1030 osnewl=&FFE7:oswrch=&FFEE
 1040 temp=&70:ptr=&71
 1050 buff=&73:len=&77
 1060 FOR pass=4 TO 7 STEP 3
 1070 O% = code:P% = &410
 1080 [OPT pass
 1090 LDX#&F2: LDY#0: LDA#1: JSR osargs
 1100 LDY#0: JSR gsinit: BNE oneonly
 1110 JMP listall
 1120 .oneonly
 1130 JSR gsread: CMP #ASC"0": BCC notno
 1140 CMP #ASC"9"+1: BCS notno
 1150 AND #&F: STA temp: JSR gsread
 1160 BCC more
 1170 JMP nok
 1180 .more
 1190 CMP #ASC"9"+1:BCS syntax
 1200 CMP #ASC"0":BCC syntax
 1210 AND #&F:PHA
 1220 LDA temp:ASL A:ASL A:ADC temp
 1230 ASL A:STA temp:PLA:ADC temp
 1240 STA temp:BRA nok
 1250 .notno
 1260 STA buff:LDX #1
 1270 .rdnam JSR gsread:BCS namend
 1280 STA buff,X:INX:BRA rdnam
 1290 .namend
 1300 STX len:LDA#vtab MOD &100:STA ptr
 1310 LDA#vtab DIV &100:STA ptr+1
 1320 CLR temp
 1330 .nxtv
 1340 LDY #0:LDA (ptr),Y:BEQ nferr
 1350 .nxtv2
 1360 CPY len:BEQ nok
 1370 LDA (ptr),Y:CMP buff,Y:BNE trynext
 1380 INY:CMP #4:BEQ nok:BRA nxtv2
 1390 .trynext
 1400 INC temp
 1410 LDA ptr:CLC:ADC #4:STA ptr:BCC nxt
v
 1420 INC ptr+1:BRA nxtv
 1430 :
 1440 .syntax
 1450 BRK:EQUB &DC
 1460 EQUS "Syntax: VECTOR (<no.>|<name>
)"
 1470 EQUB 0
 1480 :
 1490 .nok
 1500 LDX temp:CPX #27:BCC nok2
 1510 .nferr
 1520 BRK:EQUB 0:EQUS "Vector not found"
:EQUB 0
 1530 .nok2
 1540 JSR header:BRA vlist
 1550 :
 1560 .listall
 1570 JSR header:LDX #0
 1580 .liall2
 1590 PHX:JSR vlist:PLX
 1600 INX:CPX #27:BNE liall2
 1610 RTS
 1620 :
 1630 .header
 1640 JSR txtprt
 1650 EQUS "## Name Vadd Value OSadd Xad
d Value Rom"
 1660 EQUB 13
 1670 EQUS STRING$(39,"-")
 1680 EQUB 13
 1690 NOP:RTS
 1700 :
 1710 .vlist
 1720 PHX:TXA:LDX #0
 1730 .vlist2 CMP #10:BCC vlist3:SBC#10:
INX:BRA vlist2
 1740 .vlist3 PHA:LDA #ASC" ":CPX #0:BEQ
 vlist4
 1750 TXA:ORA #&30
 1760 .vlist4 JSR oswrch:PLA:ORA#&30:JSR
 oswrch
 1770 JSR spprt:PLA:PHA:ASL A:ASL A:TAX:
LDY #3
 1780 .vlist5 LDA vtab,X:JSR oswrch
 1790 INX:DEY:BPL vlist5:JSR spampprt
 1800 LDA #ASC"2":JSR oswrch
 1810 PLA:PHA:ASL A:TAX:JSR hexprt
 1820 JSR spampprt:LDA &201,X:JSR hexprt
 1830 LDA &200,X:JSR hexprt
```

Continued on page 46



Peter Rochford reports on the latest publication to provide support for Master users, The Advanced Reference Manual from Watford Electronics, priced at £19.

When the BBC Master was first released some two years ago, there was much justifiable criticism over Acorn's decision to issue only a fairly basic Welcome Guide with the machine and charge for the additional reference manuals. The additional manuals were to comprise three volumes, Part 1, Part 2 and an Advanced Reference Guide.

Part 1 and 2 appeared after several months, and satisfied the immediate needs of many who had bought a Master and wanted to get the best out of their machine. However, others like myself, still considered that much of the detailed information required by programmers and hardware designers was not available in either of these manuals.

During the last two years, no advanced guide to the Master has been available, Acorn having failed to release the third manual. Recent months have seen the release of an updated version of the old *Advanced User Guide*, (the new one containing extra information on the Master), and a book from Dabs Press called *The Master Operating System*. Both of these books were reviewed in BEEBUG Vol.6 No.6.

Now, finally, the long awaited *Advanced Reference Guide* for the Master has been published and interestingly enough, not by Acorn. Instead, Acorn gave permission to publish the 288 page spiral-bound book to Watford Electronics, a company who are well-known to the majority of Beeb users as a major supplier of Acorn equipment, and also as designers and manufacturers of numerous add-ons and software packages for the BBC micros.

The front cover of the book is very similar to that of the manuals already published by Acorn for the Master series, with the same picture of the Master and the large M logo, while the back cover carries a picture of Watford Electronics' premises, lest you forget who the publishers are. The layout and general design of the book also follows closely the same pattern as the Acorn manuals, although much thicker paper is used, probably to make the book appear longer.

MACHINE ARCHITECTURE

The book kicks off with a look at the machine architecture of the Master. This is a general overview describing the various sections of the Master's hardware and their functions.

What follows after this is a fairly detailed description of the circuitry of the Master. Included at the back of the book are copies of two circuit diagrams supplied by Acorn Computers. The material in both of these first two chapters will be familiar to anyone who has read the Acorn service manual for the Master.

The next few chapters take a look at memory organisation, the keyboard controller, the screen display, the real-time clock, the user port, serial port, peripheral bus controller and the 1MHz bus. For those who have always wanted to gain access to the alarm function of the real-time clock, there is a very informative section devoted to this, but be warned that it is only of use to those who have good machine code knowledge.

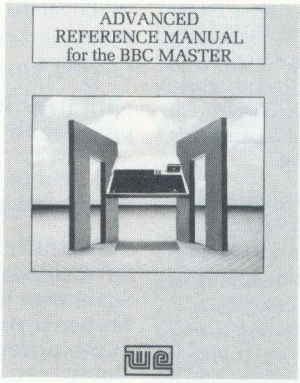
The chapter on the machine operating system mercifully has not been padded out with all the FX calls that already appear in Part 1 of the Acorn manuals. It is a relatively small section of the book, but does contain detailed information on the address map and explains how to extend the MOS.

DUAL PROCESSOR SYSTEMS

Dual processor systems are covered in great depth, and for me this is one of the highlights of the book. Tube architecture, tube protocols,

operating system usage and operating system calls are all covered. The 6502, Z80 and 80186 second processors are all covered in detail, and I found the section on the 80186 most interesting and informative.

The sections on the disc filing systems occupy only a few pages, these being given over mainly to describing the track format of the DFS, ADFS and CP/M. Quite rightly, the book points out that Acorn's *Master Reference Manual Part 1* already contains much detailed information on the two main disc filing systems.



The final few chapters of the book are devoted to the network filing system (ANFS), the Terminal Emulator, the Editor and a description of the View and ViewSheet formats.

APPENDICES

The last 120 pages of the book are split into eight appendices (i.e nearly half its length). The first three of these deal with the functional differences between the various Acorn BBC machines. These are excellent and will be extremely useful to those who need a quick reference guide when writing software and wish to make it portable across the whole range of machines.

Appendix 4 covers differences between the ANFS and the older NFS, while appendix 5 looks at the changes introduced in Basic IV.

Appendices 6 and 7 will be of most use to hardware designers as they cover PCB selection links, test points and the cartridge interface.

The final appendix is a bit of a disappointment as it takes up 67 pages of the book to list the 65C12 instruction set. The codes are already listed in the Acorn manuals, and elsewhere, and to take up one page for each code is silly. As far as I am concerned this is just padding out the book.

CONCLUSIONS

Well, should you buy this book or not? A lot depends on where your particular interest lies in relation to your Master. The book definitely contains a lot of information about the hardware of the machine. The opening chapters are particularly detailed in this area, but throughout the book too, there is much information on this subject.

Software writers will find the book useful as well though, with plenty of material to interest them. However, because of the nature of most of this material, a competent knowledge of machine code programming is essential to make use of it.

How does the new book compare with the other two referred to earlier? There is certainly some duplication of information that also appears in both the Acorn Reference manuals and the *New Advanced User Guide*. In fact, it is rather difficult to compare this book with the *New Advanced User Guide* (NAUG) and indeed Dabs Press' *Master Operating System*. The Dabs book certainly has more detailed information on the MOS as one would expect. The NAUG, however, covers many more areas than the *Advanced Reference Manual* and I think that it has greater appeal to a wider number of Master users as a general reference manual.

At the end of the day it is best to sum up by saying that *The Advanced Reference Manual* will appeal to hardware designers, software writers who need very detailed information, and those who just have an unquenchable thirst for knowledge about their Master. I found it a fascinating book to browse through and there is certainly plenty of in-depth information in it which has not until now been generally available to the public. However, I do feel that the book is over priced for what it offers.

TALKING TO EDIT

Peter Smith

Most people know how to transfer a Basic program into EDIT and back again by typing EDIT from Basic, and then pressing Shift-f4 and typing BASIC to return. To allow this to work, both Basic and Edit have a special feature for transferring data between each other, and this could be used in your own programs.

If, when Basic is invoked by *BASIC, the command is followed by an '@', (i.e. *BASIC @), then the Basic ROM will take the address in memory locations 0 and 1 as a pointer, and read the ASCII text from that address onwards, transferring it into its buffer, just as if it had been typed at the keyboard. For example, if ASC-PROG is a Basic program saved in ASCII format, then an alternative to using *EXEC to load it is to use instead:

```
*LOAD ASC-PROG E00
!0=&E00:*BASIC @
```

and the program will be read in by Basic.

EDIT SEARCH AND REPLACE

Jane Fletcher

While on the subject of Edit, here are two useful search and replace sequences for editing Basic programs:

```
$* ^#/$<Return>
```

will strip the line numbers from a Basic program, while:

```
|J/<Return>
```

will strip out the linefeeds added to each line when a listing is spooled to a file.

OVERVIEW PROBLEMS

David Spencer

Some members have written to say that the Keeper in Acorn's Overview , which is used to allow different View packages to be used simultaneously, clashes with certain other ROMs. The reason for this is that the Keeper claims the command line vector, as do some other ROMs. The solution is simply to unplug either Overview or the offending ROM, using *UNPLUG. B

VECTORING AROUND (continued from 43)

```
 1840 JSR spampprt:LDA #&FF:JSR hexprt
 1850 PLA:STA temp:TXA:ADC temp:STA temp
 1860 JSR hexprt:JSR spampprt
 1870 LDA #ASC"D":JSR oswrch
 1880 LDA temp:CLC:ADC #&9F:STA temp:TAX
 1890 JSR hexprt:JSR spampprt
 1900 LDA &D01,X:JSR hexprt
 1910 LDA &D00,X:JSR hexprt
 1920 JSR spprt:JSR spprt
 1930 LDA &D02,X:JSR hexprt2
 1940 JMP osnewl
 1950 .spprt
 1960 LDA #ASC" ":JMP oswrch
 1970 .spampprt
 1980 JSR spprt
 1990 LDA #ASC"&":JMP oswrch
 2000 .hexprt
 2010 PHA:LSR A:LSR A:LSR A:LSR A:JSR he
xprt2
 2020 PLA:AND #&F
 2030 .hexprt2
 2040 ORA #&30:CMP #&3A:BCC hexprt3:ADC
#6
 2050 .hexprt3
 2060 JMP oswrch
 2070 .txtprt
 2080 PLA:STA ptr:PLA:STA ptr+1:BRA txtp
rt3
 2090 .txtprt2
 2100 JSR osasci
 2110 .txtprt3
 2120 INC ptr:BNE txtprt4:INC ptr+1
 2130 .txtprt4
 2140 LDA (ptr):BPL txtprt2:JMP (ptr)
 2150 .vtab
 2160 EQUS "USERBRK IRQ1IRQ2CLI BYTEWORD
WRCH"
 2170 EQUS "RDCHFILEARGSBGETBPUTGBPBFIND
FSC "
 2180 EQUS "EVNTUPRTENETVDU KEY INS REM
CNP "
 2190 EQUS "IND1IND2IND3"
 2200 EQUB 0
 2210 ]NEXT
 2220 ENDPROC                          B
```

DEBUGGING DATA STATEMENTS

Peter Osborn describes his simple technique for checking the many DATA statements often found in magazine listings, including some of those in BEEBUG.

When a BBC Basic program containing many DATA statements, each with many data items, is typed into the micro from a listing, it is easy to make mistakes. The short program, COMMAS, listed here is a utility designed to help with pin-pointing any faulty DATA lines.

The output of the program is a list of the line numbers of the DATA statements, and the number of data items contained in each. Once this has been obtained, a comparison with the original listing will help to show up any discrepancies.

USING THE PROGRAM

Type in the program and save before trying it out. When run, it prompts for the name of the target program. Once this has been entered, the target program will be scanned, and output to screen and printer (if enabled) will follow.

PROGRAM NOTES

The target program is loaded at a suitable address (loadaddr%) with an OSFILE call. The end address of the loaded program is calculated from information in the OSFILE block, and a pointer to the start of the first line (pointer%) is initialised.

The REPEAT-UNTIL loop takes each line in turn and examines it for the presence of the token for the DATA statement, &DC (User Guide page 483). If it is found, PROCcount is entered. This counts the number of commas (ASCII &2C) present, and at the end of the line outputs the current line number and the number of data items apparently found (commas+1). Then pointer% is set to the start of the next line of the target program, and the process repeated until the end address of the file is reached.

```
   10 REM Program COMMAS
   20 REM Version B1.3
   30 REM Author  Peter Osborn
   40 REM BEEBUG  May 1988
   50 REM Program subject to copyright
   60 :
  100 MODE 7:VDU 15
  110 ON ERROR GOTO 300
  120 filename%=&C00:osfile=&FFDD
  130 loadaddr%=PAGE+&500
  140 block%=&70:block%?6=0
  150 !block%=filename%:!(block%+2)=load
addr%
  160 INPUT"Input the name of the file:
" A$
  170 $filename%=A$
  180 X%=&70:Y%=0:A%=255:CALL osfile
  190 pointer%=loadaddr%
  200 endaddr%=loadaddr%+256*block%?&B+b
lock%?&A-3
  210 REPEAT
  220 eoln%=pointer%+pointer%?3-1
  230 FOR J%=pointer%+4 TO eoln%
  240 IF ?J%=&DC THEN PROCcount(J%):J%=e
oln%
  250 NEXT J%
  260 pointer%=pointer%+pointer%?3
  270 UNTIL pointer%>endaddr%
  280 END
  290 :
  300 ON ERROR OFF:CLOSE#0
  310 REPORT:PRINT" at line ";ERL
  320 END
  330 :
 1000 DEF PROCcount(J%)
 1010 LOCAL I%,commas%
 1020 FOR I%=J%+1 TO eoln%-1
 1030 IF ?I%=&2C THEN commas%=commas%+1
 1040 NEXT I%
 1050 PRINT 256*pointer%?1+pointer%?2;SP
C1;commas%+1
 1060 ENDPROC
```

BEEBUG WORKSHOP USING PRINTERS (pt3)

In this, the final Workshop dealing with printers, we shall cover the use of *FX3 and *FX5, and delve into the printer itself to explain how to configure it to print in different character sets.

Inside every printer is a bank of DIP (dual in parallel) switches. There may even be two or three such banks of DIP switches, the number depending upon the complexity of the printer. Each switch will be numbered for simplicity and will have two settings, on or off. To gain access to the bank of switches you must either remove a panel (as with the Epson FX80) or remove the top of the printer (as with some Taxan printers). The most common features that these switches control are:

1. Page length.
2. Paper end detection.
3. The input buffer.
4. Slashed zero.
5. Automatic line feed.
6. Character set.

A detailed account of each feature will be given in your manual (look in the index under 'DIP switches'). There is a detector in most printers that will determine when there is no more paper. One of the DIP switches will decide whether the printer should stop printing or not when the paper has run out. Although disabling this facility may allow you to print right to the bottom of a sheet of paper, you may well run the danger of getting ink on the roller. For this reason it is wise to keep this facility enabled. Should it need to be disabled temporarily you should do this in software e.g. the KP815 code is 27,56 whilst 27,57 enables it. The page length switch must be set according to the paper being used. The choice is usually between 11 and 12 inch paper. The automatic line feed switch determines whether a Line Feed will be performed every time a Carriage Return character is received. The *FX6 command performs much the same function in software. We would recommend that this feature is disabled so that the software can decide whether a Line Feed is to be performed or not. Many printers provide a choice of two alternative characters for the number zero. In one form it will have a slash through it, in the other it won't. Set this switch according to your own taste.

Practically all modern printers incorporate some resident RAM. If this is the case with your printer, there will be a switch that will determine whether this RAM is to be used to store new character definitions, or whether it is to be used as a printer buffer. Many people do not realise that they can configure their printer to use a buffer by the mere setting of a switch, which is usually set for character definitions. If this is the case, altering this switch will activate the buffer. If you are interested in defining new characters to download to the printer, refer to the article Epson Character Definer in BEEBUG Vol.6 No.6.

Most dot matrix printers support more than one character set, the one in use being selected by up to three DIP switches. The different character sets are much the same as the default, but with variations on half a dozen or so characters. For example, if your printer is configured for the American character set it will print a hash sign instead of a pound character on receipt of the character Shift-3 (ASCII 35). The exact differences, and the character sets available, will be described in some detail in your printer manual. Note that the other

character sets can be software selected, so you can still print a pound character even with the DIP switches set to the American character set.

There are, of course, many other features that can be selected from the switches such as NLQ, disable bell etc, but these tend to be very 'printer specific' and a description of their use is best left to the appropriate printer manual. Finally, make sure that the printer is turned off before altering any of the switch settings because these are read only when the printer is turned on. Altering the switches while the printer is powered up will have no effect.

USING *FX3 AND *FX5

These two commands control the output streams to which data is sent by the computer. Because of the large variety of printers available, provision has been made for the use of both parallel and serial printers. The four output streams available are the screen, the RS423 port, the printer and spool. The *FX3 command must be followed by a single value specifying the output stream. The most useful values are in the range 0 to 11 (values 16 through to 27 do the same as 0 to 11 but turn spool off) and allow data to be sent to all or any of the output streams as shown in the table.

Value	Printer	Screen	RS423
0	enabled	on	off
1	enabled	on	on
2	enabled	off	off
3	enabled	off	on
4	off	on	off
5	off	on	on
6	off	off	off
7	off	off	on
8	on	on	off
9	on	on	on
10	on	off	off
11	on	off	on

Note that the printer may be on, off, or just enabled. If the printer is on, data will be sent to it irrespective of whether a VDU2 has been issued or not. However, if it is 'enabled' data will be sent to it only if a VDU2 (Ctrl-B) has been issued. A useful combination is *FX3,10. This will send data purely to the printer. If you are printing a document using lots of control codes, this is an easy way of making sure that these codes do not go to the screen where they could have unpredictable results. With this command there is no need to issue any VDU2 or VDU1 commands, but you will need to issue *FX3,0 to restore output to the normal default of screen only when printing is finished.

The *FX5 command deals purely with the printer output stream. There are four possible values in the range 0 to 3. *FX5,1 and *FX5,2 select output to the printer (parallel) or serial ports respectively. *FX5,3 selects a user supplied printer driver. For more details refer to the article 'Disc Spooler Utility' in this issue of BEEBUG. The *FX5,0 command selects a 'printer sink' where characters are simply 'lost'. This avoids wasting paper when testing programs that output their data to the printer, or where a printer is temporarily unavailable. The Beeb will not then 'hang' waiting for the printer to go 'on line'.

That concludes this series of Workshops on the use of printers. You should now be able to use most of the facilities that your printer offers from within Basic or any word processor. Equally important, you should now be able to extract the relevant commands for any function from your printer manual, and use these in your own programs. B

Points Arising....Points Arising....Points Arising....Points Arising....

VIDEO CATALOGUER (Vol.6 Nos.9 and 10)

Unfortunately the three additional lines included under points arising last month were incorrectly numbered to fit in with part two. To correct this, change lines 3930, 3940 and 3950 to read 3915, 3916 and 3917 respectively.

B

A FLASH UTILITY

If you find the BBC's flashing colours limited and not particularly useful, read how Colin Reynolds' utility transforms this situation, and provides a useful tool for animation.

Have you ever wished to use a flashing colour that was not black & white, red & cyan, or one of the other defaults? The flashing colour combinations and rates on the BBC micro are very basic, and users may find these limiting. This program extends flashing colours to a level where they can produce some really interesting effects if used imaginatively.

COLOUR	TIME 1/50 SECOND 0	70	85
1	RED	BLUE	
2	BLACK	RED	

Colour/Time Chart Example 1

The program allows the user to re-define any of the logical colours to flash between any two of the physical colours at any rate. You can even switch between two physical flashing colours.

HOW TO USE THE PROGRAM

Each logical colour is re-defined by seven values. To use the program, after you have typed it in and saved it, you will need to specify suitable values in DATA statements between lines 800 and 990. The meaning of these values is:

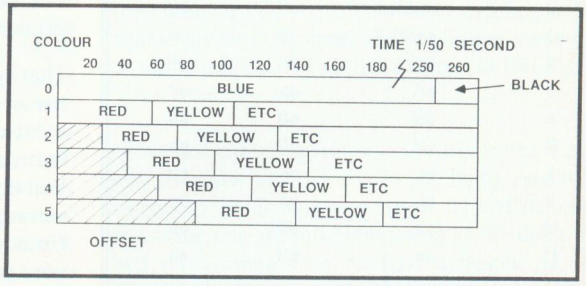
1 - Logical colour to be defined.
2 - 1st physical colour.
3 - 2nd physical colour.
4 - Period of 1st colour (50ths of a second)
5 - Period of 2nd colour (50ths of a second)
6 - Counter (usually 1)
7 - Flag (usually 0)

The sixth value can nearly always be set to 1 for each colour defined. It is however possible, by changing this value, to get special effects, as described in the second example below. The flag is used to keep track of which colour is selected, and can always be left as 0.

To re-define another logical colour just add another DATA statement with the information in it as described above. The example in the program will re-define logical colour 1 (selected by COLOUR 1 or GCOL 0,1 etc.) to flash between colours 1 & 7 (red & white), with red selected for 1 second (50/50 ths) and white for 0.4 seconds (20/50 ths). When the program is run, it will set up the new colours, and these can then be disabled with *FX13,4, and subsequently re-enabled with *FX14,4 from within your own program.

PROGRAM NOTES

The program uses the start of vertical sync event, which is enabled by *FX14,4, to generate an event 50 times a second. For each logical colour defined, a counter is decremented each time the event occurs. Once this reaches zero the next colour is selected and the counter is loaded with the 'on' time of the new colour.



Colour/Time Chart Example 2

As the program is so short (80 bytes), it can be fitted, together with enough data to re-define all 16 colours, into 200 bytes of memory. This means that it could be assembled, say, in the serial buffer at &900, and *SAVEed as a block of memory.

SCREEN DESIGNS

If you wish, you can define colours so that as one goes off another comes on, and so simple animation can be produced. This is illustrated in the examples below. To help you design these colours it is best to draw time/colour charts, and these are included for each of the following examples. To use the examples, in each case load in the the main program and then type in the example lines before running.

Example 1.

```
600 MODE 2
610 COLOUR 1:PRINT TAB(5,10);"BEEBUG"
620 COLOUR 2:PRINT TAB(8,13);"BEEBUG"
630 COLOUR 7:END

800 DATA 1,1,4,70,15,1,0
810 DATA 2,0,1,70,15,1,0
```

Example 2.

```
600 MODE2
610 FOR I=0 TO 15:GCOL 0,(I MOD 5)+1
620 PLOT 4,640,0:PLOT 5,I*80,900
630 NEXT:END

800 DATA 0,4,0,250,10,1,0
810 DATA 1,1,3,50,50,1,0
820 DATA 2,1,3,50,50,21,0
830 DATA 3,1,3,50,50,41,0
840 DATA 4,1,3,50,50,61,0
850 DATA 5,1,3,50,50,81,0
```

In colours 2 to 5 of the last example, the start counter (the 6th value in the data statement) has been set to a value other than 1. This means that although all the colours have the same flash rate, the flashing is staggered because of the different starting values. There is a lot of scope for experimentation here.

```
 10 REM Program Multi Flash
 20 REM Version B1.2
 30 REM Author  Colin Reynolds
 40 REM BEEBUG  May 1988
 50 REM Program subject to copyright
 60 :
100 DIM PROG% 200
110 OSWORD=&FFF1
120 FOR C=0 TO 3 STEP 3
130 P%=PROG%
140 [OPT C
150 .start
160 PHP:PHA:TXA:PHA:TYA:PHA
170 LDX #&00
180 .loop
190 LDA sto,X:CMP #&FF:BEQ exit
200 DEC sto+5,X:BNE next
210 TXA:CLC:ADC sto+6,X:TAY
220 LDA sto+3,Y:STA sto+5,X
230 LDA sto+6,X:EOR #&01:STA sto+6,X
240 LDA sto,X:STA spc
250 LDA sto+1,Y:STA spc+1
260 TXA:PHA
270 LDX #(spc AND 255)
280 LDY #(spc DIV 256)
290 LDA #&0C:JSR OSWORD
300 PLA:TAX
310 .next
320 TXA:CLC:ADC #&07:TAX
330 JMP loop
340 .exit
350 PLA:TAY:PLA:TAX:PLA:PLP
360 RTS
370 ]
380 spc=P%
390 $spc=STRING$(5,CHR$0)
400 P%=P%+6
410 sto=P%
420 RESTORE
430 REPEAT
440 READ A%
450 ?P%=A%:P%=P%+1
460 UNTIL A%=255
470 NEXT
480 ?&220=PROG% AND 255
490 ?&221=PROG% DIV 256
500 *FX 14,4
510 :
800 DATA 1,1,7,50,20,1,1
990 DATA 255
```

B

This month's magazine disc contains an extended demonstration, illustrating some of the effects that can be achieved with the new flashing colours.

Wrenching the controls to port caused the ship to keel over and drop sickeningly out of the purple rain-clouds. Glancing at the orward view-screen, I could make out the far mountain range piercing the shroud of the perpetual thunderstorm. The deeper hued purple of the cratered plain rushed skyward as the retro-rockets cut in, causing the ship's hull to shudder uncontrollably. Spying a small clearing lying at the foot of a large rock-strewn defile, I kicked the rudder and applied lateral thrust until the ship grudgingly lurched into the shadow of the overhang. With a final roar of triumph the engine cut and allowed the support struts to sink into the purple sands of Doom. I had returned.

Product	RETURN TO DOOM
Supplier	Topologika P.O. Box 39, Stilton PE7 3RL. Tel. 0733-244682
Price	£12.95 inc. VAT (disc only)

Having previously survived the acid rain of the planet Doom, only a fool would return, but who could ignore the distress signal of the space cruiser Galapoxi? The ship was carrying the Earth's Ambassador to Flux, and somehow it had been forced down over the dreaded planet. In the short time available before your ship is destroyed by the metal-rotting rain you must overcome the obstacles and rescue the Ambassador from her captors.

In this sequel to Countdown to Doom, Peter Killworth has managed to squeeze a quart of fun and puzzles into the BBC's pint of memory. Peter has again disregarded the opportunity to include an over-clever command parser, and instead opted for small answers to big puzzles. You will find that in general, dropping, throwing and waving are the order of the day without any superfluous subtleties. However, this does not mean that things are any the easier for the player as the problems are as convoluted as ever.

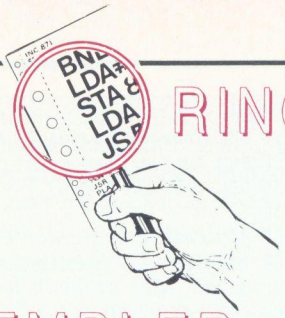
To tighten the screw, the game appears to include more than a few red herrings. Inserting objects which have no purpose is one way of confusing the player, but Peter has taken the dishonourable step one stage further. By permitting the player to use a useless object in what appears to be a useful manoeuvre, he thereby convinces the player that the object is bona fide, but in fact takes him further up the creek. In addition, the game is required to be played in the correct sequence of moves, as failure to arrive at a certain location within a set number of moves will also seal your fate.

The game does not permit you to **EXAMINE** objects, and this I personally find annoying. I have heard and appreciate Peter's argument that such fripperies are not required, but I don't agree. With a disc-based database to which this game has access, there is the space to provide a more friendly interface between the player and the adventure. If I am holding an object such as a computer and the game does not understand **EXAMINE COMPUTER, SWITCH ON COMPUTER** or **USE COMPUTER** I start to get resentful. I realise that if I bide my time, sooner or later in the game there will arrive a time where the appropriate use of the object in question will become more obvious, but before that happens a novice player will have stamped the obdurate machine into a pile of silicon chips with frustration.

Return to Doom contains all the humour you would expect from Peter, and again his 'engineering bent' lends credibility to the puzzles. There is a Montypython who will squeeze you, a Grobbler who will gobble you, and a Stereo Rock monster who will grind you up. For the faint-hearted a built-in hint facility has been included from which you may request a series of increasingly obvious clues to any problem. It should be mentioned that the game does contain a bug which surfaces if you should die at certain locations. On being asked if you wish to start a new game, the program crashes with a 'NOT FOUND' message if you

Continued on page 66

EXPLORING ASSEMBLER



Part 10

A series for beginners to machine code by Lee Calcraft

This month: Integer Division

DEDUCING AN ALGORITHM

Last month we tackled the problem of multiplication in assembler. Division, I am afraid, is no easier - though as with multiplication, dividing by powers of two is extremely easy. To divide an integer by 2, just shift it right by one place. To divide by 4, shift by two places, and so on. But for a general purpose division routine, our best bet is to begin by taking a look at longhand division, to see the exact process involved.

Take for example the number 325 divided by 14. In longhand this will take the following form:

```
                023        Quotient
Divisor    14 | 325        Dividend
                3
               14
               --
                0

               32
               14
               --
                2

               45
               14
               --
Remainder       3
```

Here we begin by taking the first digit of the dividend (3), and trying to divide the divisor into it. It will not go, so we place a zero as the leftmost digit of the quotient. The next digit of the dividend (2) is taken down and placed beside the first digit, making 32. We again see how many times 14 will divide into it. The answer this time is 2, so we insert a 2 as the second digit of the quotient. 2x14 is then subtracted from the 32, and the remainder (4) is used as the top digit of the partial dividend for the next division. This time the final digit of the dividend (5) gives a final partial dividend of 45. The result of this division is 3, and this takes its place as the final digit of the quotient. Finally we subtract 3x14 from the partial dividend to give the remainder. The final result is thus 23, with a remainder of 3.

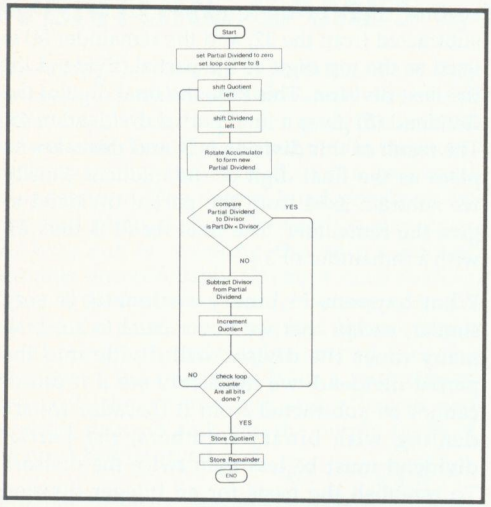
What happens in binary arithmetic is very similar, except that we never need to see how many times the divisor will divide into the partial dividend: we need only see if it can or cannot be subtracted from it (because we are dealing with binary numbers, the partial dividend must be less than twice the divisor). To establish the basis for an integer division algorithm, we will try dividing 110110 by 101 (i.e. 54 divided by 5).

```
          001010
101 | 110110
        101
        ---
        001
          111
          101
          ---
           10
          100
          101
          ---
```

The first step is to attempt to subtract the divisor (101) from the top bit of the dividend. It will not go, so we place a 0 in the top bit of the quotient, and tack on the next bit of the dividend. This partial dividend (11) is still too small, so we repeat the process. This gives us a partial dividend of 110. The divisor *can* be subtracted from this, so we perform the subtraction, and the remainder (1) drops down to take its place as the top part of the next partial dividend, and we place a 1 in the next position of the quotient.

We repeat this process until we have made the attempted subtraction a total of n times, where *n* is the number of bits in the dividend. The

result is 001010 with a remainder of 100; or in decimal terms, 54/5 gives 10 with a remainder of 4.



We can now establish five main steps in the division process, and I can do no better than to quote Leo Scanlon's extremely succinct presentation of them in his *6502 Software Design*.

1. Shift the quotient left (initially zero) to provide a (least significant) bit position for the next quotient digit.

2. Shift the dividend left, so that another bit from the partial dividend is tested.

3. Compare the divisor to the partial dividend.

4. If the divisor is less than or equal to the partial dividend, subtract the divisor from the partial dividend and enter a 1 in the quotient.

5. If any digits remain in the dividend, return to step 1.

```
Dividend (1-255) ? 233
Divisor  (1-255) ? 19
Result=            12
Remainder=          5

Dividend (1-255) ?
```

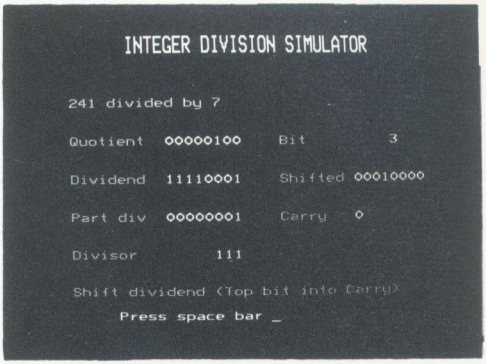
AN INTEGER DIVISION PROGRAM

A program to implement these steps is given in listing 1. When it is run, it will request a dividend and divisor in the range 1 to 255, and will perform the division, and display the result. It works as follows. The dividend and divisor are held in RAM at &70 and &71, and the partial dividend will be held in the accumulator. This is zeroed at the start (line 120), and the X register, which will hold the loop count, is set to 8. The quotient and the dividend are then both shifted one position to the left. The dividend must be shifted last, because the purpose of shifting it is to transfer its top bit to the carry flag. The accumulator is then rotated left (line 170), and as you may remember from last month, this causes the carry flag to be placed in bit zero of the accumulator, and thus forms the first bit of the partial dividend.

Listing 1

```
 10 REM Integer divide
 20 REM Author  Lee Calcraft
 30 REM Version B 0.6
 40 :
 50 dividend=&70:divisor=&71
 60 quotient=&72:remainder=&73
 70 MODE7
 80 FOR pass=0 TO 1
 90 P%=&900
100 [
110 OPT pass*3
120 LDA #0
130 LDX #8
140 .next
150 ASL quotient
160 ASL dividend
170 ROL A
180 CMP divisor
190 BCC skip
200 SBC divisor
210 INC quotient
220 .skip
230 DEX
240 BNE next
250 STA remainder
260 RTS
270 ]
280 NEXT
290 :
300 REPEAT
310 INPUT'"Dividend (1-255) ? "divid
320 INPUT"Divisor  (1-255) ? "divis
330 ?divisor=divis:?dividend=divid
340 CALL &900
350 PRINT"Result=    "?quotient
360 PRINT"Remainder= "?remainder
370 UNTIL FALSE
```

The instruction CMP divisor is then encountered in line 180. This is central to the whole process, since it checks whether the divisor will divide into the partial dividend. If it will not, then the next couple of lines are skipped. These actually subtract the divisor from the partial dividend (SBC divisor in line 200), leaving the remainder in the accumulator as the top part of the next partial dividend. The quotient is then incremented. This just places a one at the appropriate position in the quotient if division took place.



Now the loop counter is decremented (line 230), and the loop repeated until all 8 bits have been processed. Just before the routine terminates, the contents of the accumulator (which must of course hold the final remainder), are saved to *remainder* at &73.

Considering all that is involved, the routine is extremely short. But Scanlon has spotted a way to shorten it still further, and slightly increase its speed. He stores both dividend and quotient at the same location in RAM. This saves a shift operation at each of the routine's 8 cycles, and takes advantage of the fact that as the quotient builds up in RAM from the right, so the dividend is shifted out to the left.

AN INTEGER DIVISION SIMULATOR

Because of the complexity of the division algorithm outlined above, I have included a second listing which provides a demonstration of the process in action. It is written in Basic, and displays the various registers and their contents, together with the state of the carry flag during an 8 bit division. To use the program, type it in, and save it away. When it is run it will request the input of a dividend and divisor, and will then move into display mode. All the major registers will be displayed, and text indicating each step in the process will appear at the foot of the screen. At each press of the space bar, another step will be performed, and its effect will be seen on the display. This continues until the full 8 bits have been processed, and the quotient register contains the result of the division, with the remainder in the accumulator.

Next month we move on to an altogether different topic: The 6502's stack, and the use of subroutines.

Listing 2

```
   10 REM Program Division Simulator
   20 REM Version B 0.9e
   30 REM Author  Lee Calcraft
   40 REM BEEBUG  May 1988
   50 REM Program subject to copyright
   60 :
  100 MODE7
  110 PROCinit
  120 REPEAT
  130 PROCsetup
  140 PROCdivide
  150 UNTIL FALSE
  160 :
 1000 DEFPROCinit
 1010 X1=0:X2=11:X3=22:X4=31
 1020 Y$=CHR$131:C$=CHR$134
 1030 PRINT TAB(5,1)Y$CHR$141"INTEGER DI
VISION SIMULATOR"
 1040 PRINT TAB(5,2)Y$CHR$141"INTEGER DI
VISION SIMULATOR"
 1050 ENDPROC
 1060 :
 1070 DEFPROCsetup
 1080 quot=0:partd=0
 1090 VDU28,0,23,39,3:CLS
 1100 INPUT TAB(0,8)"Dividend (1-255) ",
divid
 1110 INPUT TAB(0,12)"Divisor  (1-255) "
,divis
```

```
 1120 CLS:VDU26
 1130 PRINTTAB(X1,6)C$;divid;" divided b
y ";divis
 1140 PRINTTAB(X1,9)C$"Quotient"Y$
 1150 PRINTTAB(X1,12)C$"Dividend"Y$ TAB(
X3)C$"Shifted"Y$
 1160 PRINTTAB(X1,15)C$"Part div"Y$ TAB(
X3)C$"Carry"Y$
 1170 PRINTTAB(X1,18)C$"Divisor"Y$
 1180 PROCputbin(quot,X2,9,FALSE,TRUE)
 1190 PROCputbin(divid,X2,12,FALSE,TRUE)
 1200 PROCputbin(divid,X4,12,FALSE,TRUE)
 1210 PROCputbin(0,X2,15,FALSE,TRUE)
 1220 PROCputbin(divis,X2,18,TRUE,TRUE)
 1230 ENDPROC
 1240 :
 1250 DEFPROCdivide
 1260 FOR count=7 TO 0 STEP -1
 1270 RESTORE
 1280 PRINTTAB(X3,9)C$"Bit";SPC8 Y$;coun
t
 1290 PROCtext:PROCwait
 1300 quot=FNshift(quot)
 1310 PROCputbin(quot,X2,9,FALSE,FALSE)
 1320 PROCtext:PROCwait
 1330 divid=FNshift(divid)
 1340 PRINTTAB(X4,15);ABS(carry)
 1350 PROCputbin(divid,X4,12,FALSE,FALSE
)
 1360 PROCtext:PROCwait
 1370 partd=FNrotate(partd)
 1380 PROCputbin(partd,X2,15,FALSE,FALSE
)
 1390 PRINTTAB(X4,15)"0"
 1400 PROCtext:PROCwait
 1410 IFpartd>=divis THEN success=TRUE:P
ROCtext:READA$:quot=quot+1:partd=partd-d
ivis ELSE success=FALSE:READA$:PROCtext:
READA$:READA$
 1420 PROCwait
 1430 IF success THEN PROCtext:PROCwait:
PROCputbin(quot,X2,9,FALSE,TRUE):PROCtex
t:PROCwait:PROCputbin(partd,X2,15,FALSE,
TRUE)
 1440 IFcount=0 THEN READA$
 1450 PROCtext:PROCwait
 1460 NEXT
 1470 PROCtext:PROCwait:PROCtext:PROCwai
t
 1480 SOUND 1,-15,50,2:PROCtext:PROCwait
 1490 ENDPROC
 1500 :
```

```
 1510 DEFPROCputbin(no,X,Y,suppress,fast
)
 1520 PRINTTAB(X,Y);
 1530 FOR n=7 TO 0 STEP -1
 1540 IF NOT fast THEN Z=INKEY(10)
 1550 bit=no DIV 2^n
 1560 IF bit>0 OR n=0 suppress=FALSE
 1570 IF bit=0 AND suppress=TRUE THEN bi
t=-16
 1580 VDU bit+48
 1590 no=no MOD 2^n
 1600 NEXT
 1610 ENDPROC
 1620 :
 1630 DEFPROCwait
 1640 *FX15
 1650 PRINTTAB(5,23)Y$"Press space bar "

 1660 REPEAT UNTIL GET=32
 1670 ENDPROC
 1680 :
 1690 DEFFNshift(param)
 1700 result=param*2
 1710 carry=(result>255)
 1720 =result AND 255
 1730 :
 1740 DEFFNrotate(param)
 1750 =-carry+FNshift(param)
 1760 :
 1770 DEFPROCtext
 1780 READ A$
 1790 PRINTTAB(0,21);SPC40;
 1800 PRINTTAB(0,21)C$A$
 1810 ENDPROC
 1820 :
 1830 DATA Shift quotient ready for next
 bit
 1840 DATA Shift dividend (Top bit into
Carry)
 1850 DATA Rotate part div (Shift & pick
 up carry)
 1860 DATA Compare part div to divisor
 1870 DATA Comparison succeeds
 1880 DATA Comparison fails
 1890 DATA So increment quotient
 1900 DATA and put remainder into part d
iv
 1910 DATA Repeat for next bit
 1920 DATA Division complete
 1930 DATA Quotient holds the result
 1940 DATA Remainder in partial dividend
 1950 DATA Press space for new division
```

SUPER DUMP

Turn your dot-matrix printer into a plotter and produce 'super' graphics dumps with this unique package. Geoff Bains reports.

Product	Super Dump
Supplier	Silicon Vision Signal House, Lyon Road, Harrow, HA1 2AG. Tel. 01-422 2274
Price	£15.95 inc VAT and p&p

Another printer dump at this stage in the Beeb's development has to have something pretty special to commend it. True to form, Silicon Vision's dump is both unusual in its operation and quite unique in its resulting printouts. the only other product which can compete at all is Design Dynamics Mode-00 Dump (see BEEBUG Vol.6 No.2).

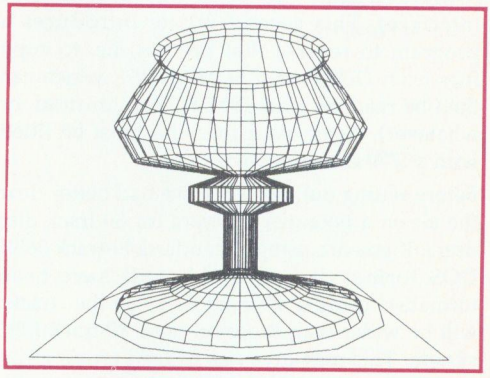
When you've created a 3D masterpiece with Silicon Vision Realtime Solids system (see the review in this issue) the last thing you want from a hard copy printout is the usual smudged, sketch of a dump from your trusty Epson. Of course, it would be nice to use a plotter but that's way beyond most people's budget. Super Dump effectively turns your ordinary dot-matrix printer into a plotter.

Like a plotter, it doesn't dump the screen image but instead uses the series of VDU commands (MOVE, DRAW, etc.) that go to make up the picture. The VDU commands are taken from a disc file and the software translates them into a high resolution image on paper. However, it won't print just one image line at a time like a plotter. It produces the image in horizontal sections like a 'normal' dump.

Not being tied to the screen means the resolution of the dump is not limited to the 640x256 which can be displayed on the Beeb, it isn't even limited to the 1280x1024 which the Beeb's screen co-ordinate system uses. In fact, the dump can be done in one of three resolutions - 640x256 (mode 0), 640x512 and a staggering 1920x1024. That's about 240 dots to the inch - almost laser printer resolution.

To produce this kind of resolution on paper, your printer must be up to it in the first place. Firstly, the printer must be Epson compatible and support a quadruple density graphics mode (which gives the 1920 dots across eight inch paper). However, the vast majority of printers can manage this. Certainly any dot-matrix printer bought today should cope. Secondly, a good ribbon is always helpful when producing high quality images.

The program is menu based; it allows the image to be scaled in either the X or Y directions and the graphics origin to be moved as well.



It's not only the Realtime Solids package which will benefit from Super Dump. Any picture which can be drawn on the Beeb's screen can be printed. You simply insert a *SPOOL <filename> into the drawing program before the drawing starts and a *SPOOL after it finishes to create a file of VDU commands. Of course, the picture must be drawn without any text or breaks to scroll or change the screen. However, most picture drawing programs can at least be altered to produce a suitable file.

Dumping pictures from commercial software is more difficult as the *SPOOL commands cannot usually be inserted into the program. Dumping from screen dump files is out of the question. Nevertheless, within these limitations, Super Dump is a marvellous piece of software. At last you can produce graphics printouts which genuinely look like their screen equivalents. B

MS-DOS TRANSFERS [pt2]

In this article Bernard Hill presents the second of two utilities to provide a means of transferring files between the Beeb and PC micros.

In last month's article we produced a program that would copy files from a disc formatted on an IBM PC to one formatted on a Beeb (provided that the Beeb has a 1770 disc interface). This second article introduces a program to reverse that process, i.e. to copy files from DFS format to MS-DOS*, which can then be read on your IBM PC (or Amstrad, or whatever). Again your machine must be fitted with a 1770 disc interface.

Before setting out, however, we had better clear the air on a potential problem for 80-track disc users. If you are using a standard 40-track 360K DOS format disc then this will have been formatted using a 40-track drive so the tracks will be wider than those on your 80-track BBC system. This may give rise to read errors when you attempt to read them on your PC's 40-track drives. This is an unavoidable hardware problem (familiar to users of both AT's and PC's) which cannot be completely overcome by software. In order to minimise this problem I have found it useful to save a file twice (the program will automatically use different names). For some reason my drives give better results when the files are on lower-numbered tracks, so use an empty MS-DOS disc.

If problems persist with your hardware combination then you will have no option other than to use 40-track drives on your BBC machine (as you will find recommended on commercially available transfer packages).

My failure rate on 80-track drives seems to be about one error in every 250K transferred, so I have been able to use my BBC at home to talk to my MS-DOS machine at work quite reliably for some weeks now.

CUSTOMISING

To convert the program for use with your system you may need to alter some of the parameters at the beginning of the program (for details see last month's article).

a. Line 140 contains a speed specification; use a value 0 to 3, 0 is the fastest, 3 the slowest drive speed.

b. The MS-DOS disc is assumed to be in drive 1 and the BBC disc in drive 0. If you have a single drive only, set D=0 in line 160 and switch discs when the program requests it.

c. Line 170 is set for 80-track drives. If you are using 40-track then change this line to read track80=FALSE.

d. The end-of-line and end-of-file conventions are different in MS-DOS and DFS. If you are transferring ASCII files then set asc%=TRUE in line 150, but if you are transferring binary data files then set asc%=FALSE. Furthermore, if your ASCII files are View files then you will need to set View%=TRUE, but if your files are spooled from any other BBC wordprocessor (such as Wordwise or EDIT) then set View%=FALSE. This is because View distinguishes between hard spaces (ASCII 32) and soft (ASCII 26) and we will need to convert to ASCII 32 for the PC.

RUNNING THE PROGRAM

When the program runs, it reads the MS-DOS directory and evaluates the free space on the disc. This may take a second or two, but eventually the menu is displayed. Options are available to catalogue the MS-DOS disc, issue a star command such as *DRIVE, *DIR or *CAT to see your BBC disc, delete a file on the MS-DOS disc (to make room for any files you wish to transfer) or, of course, to transfer a file. The file will be saved with a name which is the same as the BBC's but with the MS-DOS extension ".BBC". Should a file of this name already exist on the MS-DOS disc then it will be

saved with an extension of ".BBD" (and ".BBE" if that exists, etc.). Since a "." is not allowed in MS-DOS names then it is not possible to copy from directory X by giving a filename of "X.<file>". Instead issue *DIR X and use the short name.

If you are using a Master then the date/time stamp on the MS-DOS disc file will be taken from the system clock, otherwise it will be set to the MS-DOS default of 1-01-80. To remain consistent with last month's utility, provisions have been included to read/write the 720K discs supported by the Archimedes under the PC Emulator.

MS-DOS is a registered trade mark of the MicroSoft Corporation.

We hope that the combination of this excellent utility, along with the one published last month, will provide a quick and easy way of transferring files between the two machines. We are already finding them invaluable here at BEEBUG.

```
   10 REM Program  BBC to IBM transfer
   20 REM Version  B1.7
   30 REM Author   Bernard Hill
   40 REM Beebug   May 1988
   50 REM Program  subject to copyright
   60 :
  100 ON ERROR GOTO 610
  110 MODE7
  120 bufsiz=8*1024
  130 DIM buf% bufsiz,fat 2048,dir 3584
  140 speed=0
  150 asc%=TRUE:View%=TRUE
  160 D=1
  170 track80=TRUE
  180 A%=0:X%=1:A%=(USR&FFF4 DIV 256) AN
D &FF:Master=A%>2
  190 c$=STRING$(13," ")+CHR$134
  200 M=112:DIMsiz%(M),cl%(M),N$(M),F M+
3
  210 FOR i=0 TO 112 STEP 4:F!i=-1:NEXT
  220 L$="":PROCswitch("DOS")
  230 PROCinitpc(Master)
  240 PROCsetradr(fat):PROCgetsec(1)
  250 d2=fat?21=&F9
  260 IF d2 THEN track80=FALSE:dirsec=8
ELSE dirsec=6
  270 ns=fat!19 AND &FFFF
  280 IF ns<>720 AND ns<>1440 THEN PRINT
"Not DOS format disc":END
  290 IF fat?21=&F9 THEN maxclus=714 ELS
E maxclus=355
  300 PROCswitch("DOS")
  310 PROCtitle:PROCdirfat2
  320 dosfree=FNdosfree
  330 REPEAT:REPEAT:PROCdosfree(dosfree)
  340 f=0
  350 CLS:PRINT''TAB(13)CHR$133"OPTIONS:
"
  360 PRINT''"  ?";c$;"DOS Directory"
  370 PRINT'"  ^";c$;"Delete a DOS file"
  380 PRINT'"  *....";TAB(16);CHR$134;"I
ssue * command"
  390 PRINT'"  <filename>";TAB(16);CHR$1
34;"Transfer file to DOS"
  400 PRINT'"  RETURN/Escape";TAB(16);CH
R$129"End program"
  410 PRINT'':INPUT"        Option : "f$
  420 IF ASCf$=94 THEN PROCdelete:IF nf=
0 OR n=0 THEN 500
  430 IF f$="" THEN MODE7:END
  440 IF ASCf$=63 THEN PROCdosdir:PROCke
y:GOTO 500
  450 IF ASCf$=42 THEN PROCswitch("BBC")
:OSCLI(f$):PROCkey:GOTO 500
  460 IF nf=112 THEN PRINT'TAB(10)"DOS D
irectory full":PROCkey:GOTO 500
  470 IF INSTR(f$,".") THEN PRINT'"""."
 not allowed in filename":PROCkey:GOTO 5
00
  480 PROCswitch("BBC")
  490 f=OPENINf$:IF f=0 THEN PRINT "File
 not found":PROCkey
  500 UNTIL f<>0 : size=EXT#f
  510 IF size>dosfree THEN PRINT'"   Not
 enought room on DOS disc":PROCkey
  520 UNTIL size<dosfree
  530 PROCadddir(f$):ptr%=0:S%=0
  540 PROCswitch("BBC")
  550 REPEAT PROCxfer(BGET#f):UNTIL EOF#
f
  560 IF asc% THEN PROCxfer(26)
  570 PROCxferbuff:PROCputclus(free,&FFF
)
  580 CLOSE#f:dir!(32*ifn+28)=S%
  590 PROCdirfatback:RUN
  600 :
  610 CLOSE #0
  620 IF ERR<>17 THEN MODE7:REPORT:PRINT
" at line "ERL:END
  630 PRINT:END
  640 :
 1000 DEFPROCxfer(B%)
```

```
 1010 IF B%=26 THEN IF asc% THEN IF View
% THEN B%=32
 1020 IF B%>126 THEN IF asc% THEN ENDPRO
C
 1030 buf%?ptr%=B%:S%=S%+1:ptr%=ptr%+1
 1040 IF ptr%=bufsiz THEN PROCxferbuff
 1050 IF B%=13 THEN IF asc% THEN PROCxfe
r(10)
 1060 ENDPROC
 1070 DEFPROCxferbuff:s=PTR#f:CLOSE#f
 1080 PROCswitch("DOS")
 1090 IF ptr%<bufsiz THEN FOR I%=ptr% TO
 bufsiz:buf%?I%=0:NEXT
 1100 FOR i=0 TO (ptr%-1) DIV 1024
 1110 IF NOT first THEN last=free:free=F
Nnextfreeclus:PROCputclus(last,free):PRO
Cputclus(free,&FFF)
 1120 first=FALSE:sec=FNsecno(free)
 1130 PROCsetwadr(buf%+1024*i)
 1140 PROCputsec(sec):PROCputsec(sec+1)
 1150 NEXT:PROCswitch("BBC"):*DISC
 1160 f=OPENINf$:PTR#f=s:ptr%=0:ENDPROC
 1170 DEFPROCdosfree(n)
 1180 LOCALx,y:x=POS:y=VPOS
 1190 VDU26,31,6,3,135
 1200 PRINTFNu(n,6);" (&";~n;") DOS byte
s free"
 1210 VDU28,0,24,39,5,31,x,y:ENDPROC
 1220 DEFPROCtitle:VDU26,12
 1230 T$=CHR$132+CHR$157+CHR$131+CHR$141
+"   BEEBUG BBC to PC transfer"
 1240 PRINTT$'T$'LEFT$(T$,2)+CHR$130+"
     (Drive 0 to drive "+STR$~D+")"'CHR$
129;CHR$157
 1250 VDU28,0,24,39,5:ENDPROC
 1260 DEFFNu(v,@%):v$=STR$v
 1270 IF LENv$<@% THEN v$=STRING$(@%-LEN
v$," ")+v$
 1280 =v$
 1290 DEFPROCkey
 1300 PRINT"Any key to continue...":IF G
ET
 1310 ENDPROC
 1320 DEFPROCadddir(file$):LOCAL f
 1330 ifn=0:REPEAT c=?(dir+32*ifn)
 1340 ifn=ifn+1
 1350 UNTIL ifn>113 OR c=0 OR c=&E5
 1360 IF ifn>113 THEN full=TRUE:ENDPROC
 1370 ifn=ifn-1:free=FNnextfreeclus
 1380 IF LENfile$>7 THEN name$=LEFT$(nam
e$,7) ELSE name$=file$+STRING$(8-LENfile
$," ")
 1390 ch=ASC"C":REPEAT
 1400 new$=FNuc(name$)+".BB"+CHR$ch:f=0
 1410 REPEAT f=f+1:already=N$(f)=new$
 1420 UNTIL f>=nf OR already
 1430 IF already THEN ch=ch+1
 1440 UNTIL NOT already
 1450 PRINT'"    File saved as ";new$
 1460 $(dir+ifn*32)=FNuc(name$)+"BB"+CHR
$ch
 1470 dir?(ifn*32+11)=&20
 1480 FOR j=12 TO 21:dir?(ifn*32+j)=0:NE
XT
 1490 dir!(ifn*32+22)=FNstamp
 1500 dir!(ifn*32+26)=free
 1510 PROCputclus(free,&FFF)
 1520 full=FALSE:first=TRUE:ENDPROC
 1530 DEFFNuc(a$):LOCAL x$,a,i
 1540 FOR i=1 TO LENa$:a=ASCMID$(a$,i)
 1550 IF a>96 AND a<123 THEN x$=x$+CHR$(
a-32) ELSE x$=x$+CHR$a
 1560 NEXT:=x$
 1570 DEFPROCputsec(N):LOCAL r
 1580 PROCswitch("DOS")
 1590 T=(N-1) DIV 9:S=(N-1) MOD 9 + 1
 1600 PROCrwsec(T,S,FALSE):ENDPROC
 1610 DEFFNclus(n)
 1620 !&70=fat!(3*(n DIV 2)):!&73=0
 1630 IF n MOD 2=0 THEN =!&70 AND &FFF E
LSE =(!&71 DIV 16) AND &FFF
 1640 DEFFNsecno(c)=2*c+dirsec+3
 1650 DEFPROCdirfat2:PROCsetradr(fat)
 1660 PROCgetsec(2):PROCgetsec(3)
 1670 PROCgetsec(4):PROCsetradr(dir)
 1680 FOR s=dirsec TO dirsec+6
 1690 PROCgetsec(s):NEXT
 1700 loc=dir-32:nf=0:REPEAT loc=loc+32
 1710 t=loc?11:IF ?loc=0 OR ?loc=&2E OR
?loc=229 OR t AND &18 THEN 1740
 1720 nf=nf+1:loc?11=13:N$(nf)=LEFT$($lo
c,8)+"."+RIGHT$($loc,3):loc?11=t
 1730 siz%(nf)=loc!&1C:cl%(nf)=loc!&1A A
ND &FFFF:F?nf=(loc-dir) DIV 32
 1740 UNTIL ?loc=0 OR nf=M:ENDPROC
 1750 DEFPROCgetsec(N):PROCswitch("DOS")
 1760 PROCswitch("DOS")
 1770 T=(N-1) DIV 9:S=(N-1) MOD 9 + 1
 1780 PROCrwsec(T,S,TRUE):ENDPROC
 1790 DEFPROCrwsec(T,S,read)
 1800 *fx143,12,255
 1810 val=rst:IF D=0 THEN val=val OR 1 E
LSE val=val OR 2
 1820 IF T MOD 2=1 THEN val=val OR sel
 1830 ?ctrl=val
 1840 ?flag=1:?cmd=&C+speed:PROCwait
```

```
 1850 IF track80 THEN ?datareg=(T DIV 2)
*2 ELSE ?datareg=T DIV 2
 1860 ?flag=1:?cmd=&18+speed : REM seek
 1870 PROCwait
 1880 ?trackreg=T DIV 2:?secreg=S
 1890 ?flag=1:IF read THEN ?cmd=&84 ELSE
 ?cmd=&A6 :REM read/write
 1900 PROCwait:*DISC
 1910 ENDPROC
 1920 DEFPROCwait
 1930 REPEAT UNTIL (?status AND 1)=0
 1940 IF (?cmd AND &10)<>0 THEN PRINT"Re
ad error drive "FNu(D,1)" track "FNu(T,1
)" sector "FNu(S,1):END ELSE ENDPROC
 1950 DEFPROCinitpc(master)
 1960 IF master THEN wd=&FE28:ctrl=&FE24
:sel=16:dden=&20:rst=4 ELSE wd=&FE84:ctr
l=&FE80:sel=4:dden=8:rst=&20
 1970 cmd=wd:status=wd:trackreg=wd+1
 1980 secreg=wd+2:datareg=wd+3:S=0:T=0
 1990 ?ctrl=rst+D+1:ENDPROC
 2000 DEFPROCsetradr(a)
 2010 FOR opt=0 TO 2 STEP 2
 2020 P%=&D00:[OPT opt : PHA
 2030 LDA status: AND #1: STA flag
 2040 LDA status: AND #&1F: CMP#3
 2050 BNE exit: LDA datareg
 2060 .dest STA a: INC dest+1
 2070 BNE exit: INC dest+2
 2080 .exit PLA: RTI
 2090 .flag BRK
 2100 ]:NEXT:S=0:T=0:ENDPROC
 2110 DEFPROCswitch(a$):*fx15,1
 2120 IF D=0 AND L$<>a$ THEN PRINT"Inser
t "a$" disk : press a key":IF GET
 2130 L$=a$:ENDPROC
 2140 DEFPROCsetwadr(a)
 2150 *fx143,12,255
 2160 FOR opt=0 TO 2 STEP 2
 2170 P%=&D00:[OPT opt: PHA
 2180 LDA status: AND #&1F: CMP #3
 2190 BNE exit
 2200 .dest LDA a: STA datareg: INC dest
+1
 2210 BNE exit :INC dest+2
 2220 .exit
 2230 LDA status: AND #3: STA flag
 2240 PLA: RTI
 2250 .flag BRK
 2260 ]:NEXT:S=0:T=0:ENDPROC
 2270 DEFFNnextfreeclus:LOCAL i:REPEAT
 2280 i=i+1
 2290 UNTIL i>maxclus OR FNclus(i)=0
 2300 IF i<=maxclus THEN =i
 2310 PRINT"Cluster limit exceeded":END
```

```
 2320 DEFFNdosfree:LOCAL i,T
 2330 FOR i=1 TO maxclus
 2340 IF FNclus(i)=0 THEN T=T+1
 2350 NEXT:=T*1024
 2360 DEFPROCputclus(n,v)
 2370 !&70=fat!(3*(n DIV 2))
 2380 IF n MOD 2=0 THEN !&70=(!&70 AND &
FFFFF000) OR v ELSE !&70=(!&70 AND &FF00
0FFF) OR 4096*v
 2390 fat!(3*(n DIV 2))=!&70:ENDPROC
 2400 DEFPROCdirfatback:PROCsetwadr(fat)
 2410 PROCsetwadr(fat)
 2420 PROCputsec(2):PROCputsec(3)
 2430 IF d2 THEN PROCputsec(4)
 2440 PROCsetwadr(fat)
 2450 IF d2 THEN s=5 ELSE s=4
 2460 PROCputsec(s):PROCputsec(s+1)
 2470 IF d2 THEN PROCputsec(s+2)
 2480 PROCsetwadr(dir)
 2490 FOR s=dirsec TO dirsec+6
 2500 PROCputsec(s):NEXT
 2510 ENDPROC
 2520 DEFFNstamp
 2530 IF NOT Master THEN =&210000
 2540 X%=&70:Y%=0:A%=14:?&70=1:CALL &FFF
1
 2550 y=FNbcd(&70)-80:m=FNbcd(&71)
 2560 d=FNbcd(&72):h=FNbcd(&74)
 2570 n=FNbcd(&75):s=FNbcd(&76) DIV 2
 2580 =s+32*n+2048*h+65536*(d+32*m+512*y
)
 2590 DEFFNbcd(x)=10*(?x DIV 16)+?x MOD
16
 2600 DEFPROCdelete:LOCAL n$:PROCdosdir
 2610 IF nf=0 THEN PROCkey:ENDPROC
 2620 PRINT'TAB(10);
 2630 INPUT "Which : "n$:n=VALn$
 2640 UNTIL F?n<>255 OR n<=0 OR n>nf
 2650 IF n<=0 OR n>nf THEN n=0:ENDPROC
 2660 n=F?n
 2670 IF n=255 THEN PRINT"File not found
":ENDPROC
 2680 dir?(32*n)=&E5:c=dir?(32*n+26)
 2690 REPEAT v=FNclus(c):PROCputclus(c,0
)
 2700 c=v: UNTIL c>&FF7
 2710 PROCdirfatback:RUN
 2720 DEFPROCdosdir:LOCAL i:CLS
 2730 IF nf=0 THEN PRINT''TAB(16);"No fi
les"'':ENDPROC
 2740 FOR i=1 TO nf
 2750 PRINTFNu(i,3);TAB(10);N$(i);TAB(25
);FNu(siz%(i),6)
 2760 NEXT:PRINT:ENDPROC
```

```
char string[], part1[], part2[];
{
int c, n=0, m=0;
while((c=string[n++]) != ' ' && c != '\n' && c
!= '\0' && c != ',')
   part1[m++]=c;
part1[m] = '\0';
m = 0;
if(string[n] == '\0'){
  for(m = 0; m <=6 ; m++)
    part2[m]='\0';
  }
  else
  {
  while((c=string[n++]) && c != '\n' && c !=
'\0')
     part2[m++]=c;
  part2[m] = '\0';
  }

/* test part1, return TRUE if numeric */
n = TRUE;
for(m = 0;part1[m] != '\0';m++){
   n = n & (part1[m] >= '0' && part1[m] <=
'9');
   }
if (m = 0)
   n=FALSE;
return(n);
}

/* INSERT */
insert(linen,line)
int linen;
char line[];
{
struct txtcontrol *ptr;
struct txtcontrol *ptr1;
int n;
char c;

/* Check for null line (delete)  */
ptr1 = firstpointer;
if(line[0] == '\0'){
        /* null input - delete line */
    while(ptr1->linenum < linen && ptr1->next
!= NULL){
       ptr1 = ptr1->next;
       }
    if(ptr1->linenum == linen){
      if(ptr1->prev != NULL){
        ptr1->prev->next = ptr1->next;
        }
      ptr1->next->prev = ptr1->prev;
      if(ptr1 == firstpointer){
        firstline = ptr1->next->linenum;
        firstpointer = ptr1->next;
        }
      if(ptr1 == lastpointer){
        lastline = ptr1->prev->linenum;
        lastpointer = ptr1->prev;
        }

      return(NULL);
      }
return(NULL); /* null input, no such line */
}

/* non-null line */
ptr = &(info[nfs++]);  /* Next free cell in
info */

/* check for first/last line */
if(linen == firstline){
  ptr->linenum = linen;
  ptr->next = firstpointer->next;
  ptr->prev = NULL;
  ptr->next->prev = ptr;
  firstpointer = ptr;
  }
if(linen == lastline){
  ptr->linenum =linen;
  ptr->next = NULL;
  ptr->prev = lastpointer->prev;
  ptr->prev->next = ptr;
  lastpointer = ptr;
  }
if(linen < firstline){
  ptr->linenum = firstline = linen;
  ptr->prev = NULL;
  ptr->next = firstpointer;
  firstpointer = ptr;
  ptr->next->prev = ptr;
  }
if(linen > lastline){
  ptr->linenum = lastline = linen;
  ptr->prev = lastpointer;
  lastpointer = ptr;
  if(ptr > info)
    ptr->prev->next = ptr;
  }

/* line to be inserted between 1st and last */
if(linen > firstline && linen < lastline){
  while(ptr1->linenum < linen && ptr1->next !=
NULL)
```

```
        ptr1 = ptr1->next;
  if(ptr1->linenum == linen){
     ptr->prev = ptr1->prev;
     ptr->next = ptr1->next;
     ptr->prev->next = ptr;
     ptr->next->prev = ptr;
     }
  else{ /* inserting a new line */
     ptr->prev = ptr1->prev;
     ptr->next = ptr1;
     ptr1->prev->next = ptr;
     ptr1->prev = ptr;
     }
}
ptr->linenum = linen;
ptr->ptext = pfree;

/* info pointers now set up */
/* copy line into text      */
n=0;
while((c=line[n++]) != '\0'){
   *pfree = c;
   pfree++;
   }
*pfree = '\0';
pfree++;
}

/* INITIALISE */
/*  reset all pointers */
initialise(){
int n;
struct txtcontrol *ptr;
ptr = info;
for (n = 0; n <= LINEMAX; n++){
    ptr->prev = ptr->ptext = ptr->next = NULL;
    info[n].linenum = 0;
    ptr++;
    }
pfree = text;
firstline = 32000;
lastline = 0;
firstpointer = info;
lastpointer = NULL;
nfs = lineno = 0;
}

/* PROGLOAD */
progload(filename)
char filename[];
{
int c, sub, n;
char inline[80];
if((in=fopen(filename,"r")) == NULL){
   printf("can't open file!");
   return(NULL);
   }
lineno = 10;
sub = n = 0;
while((c=fgetc(in)) != EOF){
   if(c != '\n' && c != EOF)
     inline[n++] = c;
   else{
     if(n == 0)
       inline[n++] = ' ';
     inline[n] = '\0';
     insert(lineno,inline);
     lineno += inc;
     n=0;
     }
   }
fclose(in);
/* list first 21 lines */
proglist(10,210);
}

/* PROGSAVE */
progsave(filename)
char filename[];
{
int c,n;
char *textptr;
struct txtcontrol *ptr;
/* check if ok to overwrite existing file */
if((out=fopen(filename,"r")) != NULL){
   fclose(out);
   if(!(c=confirm()))
     return(NULL);
   }
if((out=fopen(filename,"w")) == NULL){
   printf("unable to open file!!");
   return(NULL);
   }
ptr = firstpointer;
n = 0;
while(ptr != NULL && (textptr = ptr->ptext) !=
NULL){
 fputs(textptr,out);
 fputc('\n',out);
 ptr = ptr->next;
 }
fclose(out);
}

/*  CONFIRM  */
confirm(){
int c;
printf("Do you wish to overwrite this file?
Y/N");
while((c=getchar()) != 'y' && c != 'n' && c !=
```

```
'Y' && c != 'N');
if(c=='Y' || c == 'y')
  return(TRUE);
else
  return(FALSE);
}

 /* PROGLIST */
proglist(n1,n2)
int n1;
int n2;
{
int n = 0;
char *tptr;
struct txtcontrol *pt;
pt = firstpointer;
if(n1 <= 0){
  n1 = firstline;
  n2 = lastline;
  }
while(pt->linenum < n1){
    pt = pt->next;
    }
if(n2 == 0 || n2 < n1){
  n2=lastline;
  }
do{
  tptr = pt->ptext;
  if(pt->ptext != NULL){
   printf("%4d. %s\n",pt->linenum, tptr);
    pt = pt->next;
    }
  else{
    n2 = 0;
    }
} while(pt->linenum <= n2 && n2 > 0 && pt !=
NULL);
}

/* Convert a string to an integer */
atoj(str)
char *str;
{
int x=0;
int ans =0;
if(str[0] == '\0')
   return(0);
for(; str[x] != '\0'; x++)
   ans = (10 * ans) + (str[x] - '0');
return(ans);
}
/* Put zero in null strings */
editcheck(s1,s2)
char s1[];
char s2[];
{
if(*s1 == '\0')
  *s1 = '0';
if(*s2 == '\0')
  *s2 = '0';
}
```

B

MULTI-COLUMN PAGE PRINTER (continued from page 11)

```
 3520 .cnoc DEX:BNE cloop1
 3530 RTS
 3540 :
 3550 .makeline LDY #S%:LDX bufptr
 3560 .charloop JSR osbget:BCS eof
 3570 CMP #ASC("}"):BEQ eop
 3580 CMP #ASC("{"):BEQ eoc
 3590 CMP #&0D:BEQ eol:CMP #ASC" ":
BCS notc:LDA #ASC" "
 3600 .notc STA buffer,X:INX:CPX #cwid%
 3610 BEQ charloop:BCC charloop
 3620 DEX:CMP #&20:BEQ eol
 3630 JSR split:JMP storebuffer
 3640 .eof LDA #&FF:STA eofflg
 3650 .eop LDA #&FF:STA eopflg
 3660 .eoc LDA #&FF:STA eocflg
 3670 .eol LDA #&FF:STA eolflg
 3680 LDA #0:STA bufptr
 3690 DEX:CPX #&FF:BEQ return
 3700 .storebuffer TXA:TAY
```

```
 3710 .stloop LDA buffer,Y:STA (lbase),Y
 3720 DEY:CPY #&FF:BNE stloop
 3730 .return RTS
 3740 :
 3750 .sploop DEX
 3760 .split LDA buffer,X
 3770 CMP #&20:BEQ spc:CPX #0:BNE sploop
 3780 LDX #cwid%
 3790 .spc DEX:RTS
 3800 :
 3810 .bufferreset LDY #0
 3820 .buloop INX:LDA buffer,X
 3830 CMP #&20:BEQ next
 3840 STA buffer,Y:INY:STY bufptr
 3850 .next CPX #cwid%:BCC buloop
 3860 RTS
 3870 :
 3880 ]:NEXT
 3890 ENDPROC
```

B

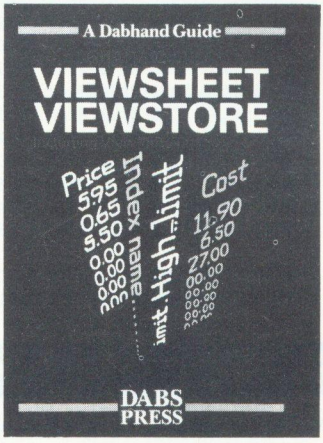
VIEW SHEET & STORE

Peter Rochford gives an overview of the latest Dabhand guide to Acorn's View family.

When I reviewed Acornsoft's ViewStore database management ROM in BEEBUG Vol.4 No.5, I was full of praise for this excellent piece of software. I did, however, point out that to get the full power from this package, you needed to study the manual very carefully. Perhaps I should have been more forthright and said that the manual was not as easy to follow as it should have been and lacked in detail in certain areas.

This criticism I think would also apply to the manuals that accompany the rest of the View family. The View productivity software suite is indeed excellent and powerful, but the manuals are neither comprehensive, nor easy to follow, particularly for those new to computing.

Dabs Press has not been slow to seize on this long-standing weakness, and following on from its initial Dabhand Guide to View, it has now released a guide to both ViewSheet and ViewStore in one book.



ViewSheet & ViewStore.
A Dabhand Guide by Graham Bell, published by Dabs Press at £12.95.

The Dabhand Guide to ViewSheet and ViewStore is a 340 page spiral-bound book aimed at those who want to get the best from their View database and spreadsheet. It is written by Graham Bell who is the editor of Acorn User.

The book is a complete tutorial and reference guide to both software packages, and contains several examples of setting up and using a database and a spreadsheet. A number of very useful utility programs are included, and these can be purchased on disc at a cost of £7.95 if you do not want the chore of typing them in.

Apart from clarifying and expanding on the information in the original manuals, the book provides a host of hints and tips, which make using both pieces of software that much simpler and quicker. Also provided are some excellent quick reference guides to both commands and error messages.

To give a detailed breakdown of the matter covered by this book is not really within the scope of this review, as there is just so much information on both pieces of software. Suffice to say that it is very comprehensive indeed, and certainly provides answers to the many questions left by the original manuals. As an example, the REPORT utility in ViewStore is one which has caused problems to many, myself included, as the detail in the original manual was so vague. This book has a large section devoted to this particular area. By use of several examples it unveils the mysteries and enables the setting up of some very complex reports with relative ease.

The book is written in an easy to follow style and should find favour with both newcomers and old hands. There are numerous examples which help to illustrate the areas covered. In addition to the information on ViewSheet and ViewStore, the author has provided details of ViewPlot and the OverView package for the Master.

In conclusion, I can find little to criticise in this book. It is well-written, instructive and informative, and provides all the right kind of information that users of ViewSheet and ViewStore are likely to need. Highly recommended. B

BEEBUG MINI-WIMP (Continued from page 23)

```
8820:C8B1 F229 DFC9 57D0 F0D9
8828:29A9 6985 77A9 8885 A796
8830:78A0 00B1 7720 E3FF 9BEF
8838:C8D0 04E6 78A0 00AA E33C
8840:D0F1 4CA2 80A2 00BD E80D
8848:5588 20E3 FFE8 C90D C910
8850:D0F5 4C9B 800A 4D49 7A14
8858:4E49 5749 4D50 2052 70C4
8860:4F4D 2056 312E 3030 5156
8868:0D0A 4D49 4E49 5749 FABE
8870:4D50 2052 4F4D 2056 90FB
8878:312E 3030 2028 4329 7F79

8880:2044 414A 2031 3938 9A1B
8888:370D 0D20 204D 5753 633A
8890:4554 5550 0D20 204D EC8E
8898:574B 4559 0D20 204D C25B
88A0:5753 5449 434B 0D20 9D3E
88A8:204D 574D 4F55 5345 3D71
88B0:0D0D 5769 6E64 6F77 0D73
88B8:733A 0D0D 2020 4D57 DF0B
88C0:4F50 454E 203C 6C78 9148
88C8:2C62 792C 7278 2C74 9208
88D0:793E 0D20 204D 5753 F3CE
88D8:4855 540D 0D49 636F 6467
88E0:6E73 3A0D 0D20 204D 1CC9
88E8:5749 434F 4E20 3C30 7E83
88F0:2D36 333E 0D20 204D 1B82
88F8:5744 4546 203C 6963 D039

8900:6F6E 206E 6F2E 2026 C0A5
8908:2033 3220 6279 7465 6B87
8910:2064 6566 696E 6974 F769
8918:696F 6E3E 0D20 204D D12E
8920:574C 4F41 4420 3C66 339C
8928:696C 656E 616D 653E 7360
8930:203C 7374 6172 7420 EB86
8938:6963 6F6E 3E0D 2020 7388
8940:4D57 5341 5645 203C 7A88
8948:6669 6C65 6E61 6D65 07BA
8950:3E20 3C73 7461 7274 5BE8
8958:3E20 3C65 6E64 3E0D 3E79
8960:0D50 6F69 6E74 6572 D590
8968:3A0D 0D20 204D 5743 785F
8970:4F4F 5244 203C 302D 4C94
8978:3339 3E20 3C30 2D33 38C3

8980:303E 0D20 204D 5750 C130
8988:4F49 4E54 4552 0D0D 76EF
8990:4E65 7720 6572 726F BF92
8998:7273 3A0D 0D20 2032 EFAD
89A0:3132 2D54 6F6F 206D 3F12
89A8:616E 7920 7769 6E64 AEBC
89B0:6F77 730D 2020 3231 C1A8
89B8:332D 4E6F 2077 696E 0F9E
89C0:646F 7720 6F70 656E C051
89C8:0D20 2032 3134 2D42 5FC3
89D0:6164 2070 6172 616D 69C1
89D8:6574 6572 730D 2020 0A8A
89E0:3231 352D 4261 6420 1D9D
89E8:6963 6F6E 0D00 5853 0886
89F0:482E 0D58 4C2E 330D 5B3A
89F8:5848 492E 0D19 0000 82D8

8A00:0000 0000 D454 6F6F 0A5C
8A08:206D 616E 7920 7769 4A61
8A10:6E64 6F77 7300 0000 0313
8A18:D54E 6F20 7769 6E64 D91E
8A20:6F77 206F 7065 6E00 98B2
8A28:0000 0000 D642 6164 79C8
8A30:2070 6172 616D 6574 B554
8A38:6572 7300 0000 0000 0C25
8A40:D742 6164 2069 636F 8605
8A48:6E00 0000 0000 0000 3115
8A50:0000 0049 434F 4E20 F103
8A58:2020 204F 5045 4E20 C807
8A60:2020 2053 4554 5550 FFF9
8A68:2020 2043 4F4F 5244 7815
8A70:2020 2050 4F49 4E54 D0C4
8A78:4552 2053 4855 5420 39BB

8A80:2020 2053 5449 434B 81B4
8A88:2020 204B 4559 2020 B9AA
8A90:2020 204D 4F55 5345 AC3A
8A98:2020 2044 4546 2020 847A
8AA0:2020 204C 4F41 4420 66A2
8AA8:2020 2053 4156 4520 C334
8AB0:2020 20AB 80BC 8117 645D
8AB8:843C 8475 841D 86D6 1400
8AC0:86DD 86E4 86EB 8649 E5BF
8AC8:8786 8707 0F1F 3F7F E3DA
8AD0:FFFF FFFF 3C3C 3838 FAC5
8AD8:0000 0000 0206 0E1E 3106
8AE0:3E7E 7E08 0810 1000 18FB
8AE8:0000 0000 0000 0000 E7B3
8AF0:0000 0000 0000 0000 4365
8AF8:0000 0000 0000 0000 D0C8
```

ADVENTURE GAMES (continued from page 52)

if you wish to start a new game, the program crashes with a 'NOT FOUND' message if you reply 'Y'. Generally speaking the disc handling routines are unfriendly. When attempting to reload a saved game there is no way to obtain a list of the saved files from within the program, and should you find yourself in such a position, there is no elegant way to recover. Equally frustrating, the program will crash if you mistakenly confirm that the program disc is in the drive when in fact your saved game disc is still resident.

This game is one for the experienced player. Individually the puzzles are not too difficult but when woven into a web such as this you will need to be prepared to restart continually from square one, and re-examine your logic repeatedly to ensure that all your basic assumptions are correct.

ARCHIMEDES NEWS

A piece of news which should be of interest to Archimedes owners is that Robico Software are at present putting the finishing touches to the upgraded version of 'Enthar Seven'. This S.F. adventure was a great favourite of mine on the BBC, and as the new version comes complete with graphics and an updated parser, I'm sure it will be a great hit on the wonder machine.

POSTBAG

POSTBAG

EXPANDING THE COMPACT

Can you please tell me if there are any ROM expansion boards, either internal or external, which are suitable for the Master Compact. I have already fitted the Mertech Compact Companion to the expansion socket so I am wondering if its bus connection is any use.

S.B.Birks

We do not know of any ROM board for the Compact, but an alternative solution is the Viglen Cartridge system - see Postbag Vol.6 No.8.

USING SHADOW RAM

I have a query concerning the shadow RAM used on the B+ and the Master. I have a font designer program which I have written which uses normal RAM to store the font, and then calls OSWRSC and OSRDSC (OSRDRM) to read and write to the screen. Although this works, on the Master the operations are much slower with a noticeable flickering. Can you tell me if it is possible to read and write shadow RAM whilst viewing a normal screen. This would give me 20K of storage space whilst being able to read and write to normal memory very quickly. Andrew Fletcher

*The answer, at least as far as the Master is concerned lies in the call FX108. This call allows you to select whether peeking and poking addresses between &3000 and &7FFF accesses main or shadow memory. *FX108 will access main memory, while *FX108,1 will access shadow memory. Therefore, putting *FX108,1 before you access the screen, and following it with *FX108, will let you read and write the shadow screen directly. Incidentally, for details of other shadow screen FX calls see the hint on page 45 of BEEBUG Vol.5 No.5. We hope to deal with this topic more fully in the next issue.*

512 CO-PROCESSORS UNCOVERED

In the March issue of BEEBUG (Vol.6 No.9) you asked what readers would like to see in the magazine. I was pleased to see the article on C, and have also found the Master pages very useful.

But there is never anything on the 512 co-processor. For instance, better and cheaper versions of C may be used with this but there was no mention of this in your C article. Your readers could be reminded that Acorn has issued a second version of the operating system software for the 512. Some help as to which Basics can be used would be useful.

I have subscribed to BEEBUG since issue 1, and I feel that if it is to meet its claim of supporting serious users, SOME help should be given to 512 users.

Mrs.E.M.Kenward

It is true that we have largely ignored the 512 co-processor in the past, as we felt this to be very much a minority interest and not within the normal ambit of BBC micro users. However, Mr. Michael Nyman of Birmingham and Mr.S.J.O'Donnell of Cornwall both wrote in similar vein supporting the 512 co-processor and seeking more support for this system from BEEBUG, and as a result we are now investigating an article on this subject. Mr O'Donnell also writes further:

C AND OTHER LANGUAGES

In your "Jottings" you ask for comments about the wisdom of running a series on programming in C. One of the prime advantages of C is its transportability and therefore its independence of machine or hardware. I wonder about the worth of an extended series on such a topic, although I would certainly encourage a short series on this or any other language. Could we perhaps have one on Pascal as well?

S.J.O'Donnell

Others have also written to indicate their approval of our series on C, and to date no voice has been raised in dissent. We do not feel, though, that Pascal warrants the same level of coverage at this time, and we have no immediate plans for any articles on this language.

B

NEAT LISTINGS

Wayne Johnson

If you dislike the '>' prompt which is inevitably printed at the end of Basic program listings, there is an easy way to prevent this. List the program as normal, using Ctrl-B to turn the printer on, but when the listing is finished press Ctrl-A followed by the Delete key, before using Ctrl-C to turn the printer off. This little trick works by sending a delete character directly to the printer, which 'rubs out' the prompt before it is printed. This will only work on printers that buffer a whole line before printing, but this includes most dot matrix printers. The Ctrl-A makes the operating system send the delete character to the printer rather than the screen.

HIGHLIGHTING A VIEW PROBLEM

Mandy Dunn

The majority of printer drivers used with the View word processor, including those supplied by Acorn, cancel any highlights that are in operation when the end of each line is reached. This can cause a real problem if you don't spot what has happened, because the second highlight code, which is supposed to turn the highlight off, will then turn it back on again. Obviously, line breaks can change while text is being edited, so special attention must be paid to highlights before printing the text. A further problem with highlights is that on version 1.4 of View (the original version), highlight codes are treated as individual characters, which upsets formatting.

Finally, most printer drivers reset the printer just before printing a document, which means that it is impossible to set up a special printing mode beforehand, as it will be cleared as soon as printing starts. The only way around this is to use a printer driver that allows the required effects to be set from within the text.

DISABLING THE ADFS

David Spencer

Many people using a model B with a 1770 disc interface, or a B+, complain that if both the ADFS and the DFS are present then PAGE is set too high to run many Basic programs. However, as long as both the DFS and ADFS are not needed at the same time, it is possible to 'remove' one of them in software, and claim back some memory. To do this, you must first find out which ROM sockets the ADFS and DFS are in. If you have a utility ROM with a *ROMS or *ROMLIST command then this is easy, otherwise you must look inside the machine to determine this. Once you know where the ROMs are, one of them can be unpluged by entering ?&DFx=-1 and pressing Ctrl-Break, where x is the single digit ROM number in hex. For example, if the ADFS is ROM 14, it can be unplugged with ?&DFE=-1, (14 being E in hex). Once disabled, the ROM remains inactive until the machine is turned off, or the appropriate location is reset to zero, e.g. ?&DFE=0, and Ctrl-Break is pressed to reclaim the workspace. Alternatively, the *PANEL command on the BEEBUG Master ROM can be used to unplug and insert ROMs.

DISC WRITE PROBLEMS

James Francis

Attempting to save a program using Acorn's 1770 DFS will sometimes result in the error message 'Disc read only'. This does not always mean that the disc is write protected, but can arise if a 40 track disc is used in an 80 track drive, even though the command *DRIVE 0 40 has been issued. The reason is that the 1770 DFS will not allow any write operations on a disc which is being double stepped. The only solution is to use a 40 track drive, or a 40/80 track drive switched to 40 track mode. You will also need to restore single stepping mode with *DRIVE 0 80. B

In response to the many requests for help we have received since reviewing Cambridge Computer's Z88 in BEEBUG Vol.6 No.7, David Spencer presents some hints and tips to help users.

BBC BASIC

There are some subtle differences between the Z88 version of BBC Basic, and the genuine article.

The first of these is that if a procedure or function takes parameters then there should be no space between the name and the opening bracket on the Z88. For example:

```
PROCaverage (total,count)
```

is valid on BBC micros, but on a Z88 the space must be removed to give:

```
PROCaverage(total,count)
```

The second quirk concerns the nesting of FOR-NEXT and REPEAT-UNTIL loops, and functions and procedures. As most users are aware, it is very bad practice to jump out of half completed loops, but the BBC micro lets you get away with such bad habits. However, the Z88 is less forgiving and will give an error message if any loops are badly nested. For example, consider the following:

```
  10 DEF PROCnaughty
  20 FOR count = 1 TO 10
  30 IF count = 20 THEN
PRINT "Line 30":ENDPROC
  40 NEXT count
  50 PRINT "Line 50"
  60 ENDPROC
```

If this procedure is called on either the Z88 or BBC it works fine, producing the output 'Line 50'. However, if the 20 in line 30 is changed to a 5, and the procedure called again, then 'Line 30' is printed, but the program then crashes on the Z88. This is because the procedure attempts to exit before the FOR-NEXT loop has properly terminated. The answer is to keep your programs well structured.

RECHARGEABLE BATTERIES

While the Z88 uses very little battery power when compared to other portables, it still doesn't come cheap when you have to spend about £2.00 on batteries for every twenty hours of use. Many people have turned to using rechargeable Nickel Cadmium (Ni-Cad) batteries instead. However, there are problems with such a move. Firstly, Ni-Cads provide only 1.2V per cell, rather than the 1.5V of ordinary batteries. With 4 batteries in the Z88 this means that the overall voltage is down by over a volt when Ni-Cads are used. The Z88 will quite happily work at this lower voltage, but it does mean that the low battery indicator soon comes on, making it impossible to judge accurately when the batteries are dangerously low. Secondly, when Ni-Cads start to flatten they do so very quickly, which means that if the Z88 starts to play up one evening, then by the next morning all the stored data may have been lost - so be warned. Incidentally, the batteries have to be recharged in a separate charger; they don't charge up when the external mains adaptor is plugged in.

CAPS LOCK

The Caps Lock on the Z88 can be set to produce lower case when Shift is held down. For example the A key on its own produces an 'A', but with Shift pressed produces an 'a'. To set this mode press Caps Lock while holding down the Square key. The Caps symbol on the display changes to lower case, and the feature remains in operation if the Caps Lock is turned off and on again. To go back to normal Caps Lock operation press the Caps Lock key while holding down the Diamond key.

DISPLAY EFFECTS

By using a simple VDU command from Basic, it is very easy to obtain effects on the LCD display such as bold and flashing. The various effects are all turned on by VDU1 followed by a character, and turned off again by the same sequence. The possible effects, which can be mixed if needed, are:

`VDU 1,ASC"B"`	Bold on/off
`VDU 1,ASC"C"`	Cursor on/off
`VDU 1,ASC"F"`	Flashing text on/off
`VDU 1,ASC"G"`	Grey text on/off
`VDU 1,ASC"R"`	Reverse video on/off
`VDU 1,ASC"T"`	Tiny font on/off
`VDU 1,ASC"U"`	Underlining on/off

For example:

```
VDU1,ASC"B",1,ASC"U":
PRINT"HELLO WORLD":VDU1,
ASC"B",1,ASC"U"
```

will produce the words "HELLO WORLD" in bold, underlined text.

BEEBUG MEMBERSHIP

Send applications for membership renewals, membership queries and ' 'ors for back issues to the address below. All membership fees, including overseas, should be i pounds sterling drawn (for cheques) on a UK bank. Members may also subscribe to RISC User at a pecial reduced rate.

BEEBUG SUBSCRIPTION RATES

		BEEBUG & RISC USER
£ 7.50	6 months (5 issues) UK only	£23.00
£14.50	1 year (10 issues) UK, BFPO, Ch.I	£33.00
£20.00	Rest of Europe & Eire	£40.00
£25.00	Middle East	£44.00
£27.00	Americas & Africa	£48.00
£29.00	Elsewhere	

BACK ISSUE PRICES

Volume	Magazine	Cassette	5"Disc	3.5"Disc
1	£0.40	£1.00	-	-
2	£0.50	£1.00	-	-
3	£0.70	£1.50	£3.50	-
4	£0.90	£2.00	£4.00	-
5	£1.20	£2.50	£4.50	£4.50
6	£1.30	£3.00	£4.75	£4.75

All overseas items are sent airmail. We will accept official UK orders for subscriptions and back issues, but please note that there will be a £1 handling charge for orders under £10 which require an invoice. Note that there is no VAT in magazines.

FURTHER DISCOUNTS

We will allow you a further discount:

Five or more:	deduct £0.50 from total
Ten or more:	deduct £1.50 from total
Twenty or more:	deduct £3.50 from total
Thirty or more:	deduct £5.00 from total
Forty or more:	deduct £7.00 from total

POST AND PACKING

Please add the cost of p&p:

Destination	First Item	Second Item
UK, BFPO + Ch.I	40p	20p
Europe + Eire	75p	45p
Elsewhere	£2	85p

BEEBUG
Dolphin Place, Holywell Hill, St.Albans,
Herts. AL1 1EX
Tel. St.Albans (0727) 40303
Manned Mon-Fri 9am-5pm
(24hr Answerphone for Connect/Access/Visa orders and subscriptions)

BEEBUG MAGAZINE is produced by BEEBUG Ltd.

Editor: Mike Williams
Assistant Editor: Kristina Lucas
Technical Editor: David Spencer
Technical Assistant: Lance Allison
Production Assistant: Yolanda Turuelo
Membership secretary: Mandy Mileham
Editorial Consultant : Lee Calcraft
Managing Editor: Sheridan Williams

CONTRIBUTING TO BEEBUG PROGRAMS AND ARTICLES

We are always seeking good quality articles and programs for publication in BEEBUG. All contributions used are paid for at up to £40 per page, but please give us warning of anything substantial that you intend to write. A leaflet 'Notes to Contributors' is available on receipt of an A5 (or larger) SAE.

Please submit your contributions on disc or cassette in machine readable form, using "View", "Wordwise" or other means, but please ensure an adequate written description is also included. If you use cassette, please include a backup copy at 300 baud.

In all communication, please quote your membership number.

Printed by Head Office Design (0782) 717161 ISSN - 0263 - 7561

Magazine Disc/Cassette

MAY 1988 DISC/CASSETTE CONTENTS

BOXED IN THE CARPARK - a delightful little game based on an original over 40 years old.

MULTI-COLUMN PAGE PRINTER a utility to allow any text to be printed in multiple column format.

FILE HANDLING FOR ALL - three programs to create, display and update serial files, plus a sample data file to experiment with.

NOW C HERE (Part 3) - a short file append program and a complete text editor both in C.

BEEBUG MINI-WIMP - both the source code, and the ROM image ready to run, for our new MiniWimp series.

DISC SPOOLER UTILITY - divert any of your printer output to disc for subsequent editing.

FIRST COURSE

CHARACTER CONTROL three short programs providing full demonstrations of this month's string handling techniques.

THE MASTER SERIES

VECTORING AROUND - a demonstration of the use of extended vectors, and a helpful utility.

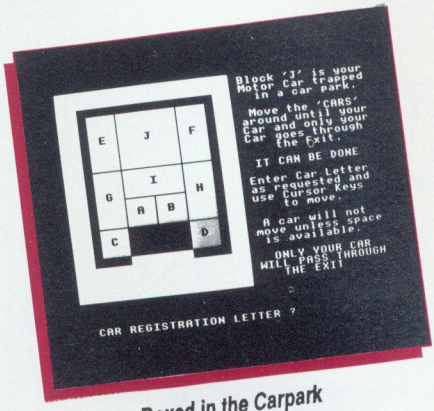
DEBUGGING DATA STATEMENTS - use this short program to check the entry of DATA statements.

the magazine program plus an extended demonstration of the control of flashing colours.

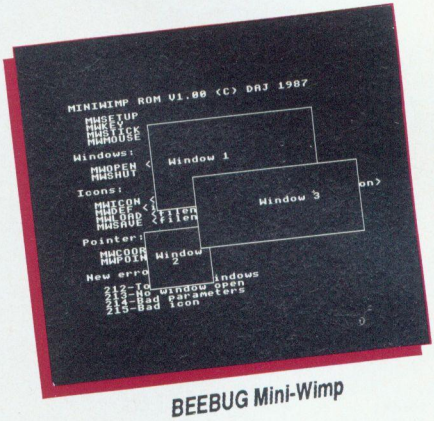
EXPLORING ASSEMBLER (PART 10) - a machine code program for integer division, and a simulation of the whole process.

IBM TO BBC TRANSFER UTILITY (Part 2) - this month the complementary utility for BBC to PC transfers.

MAGSCAN - the complete index for volume 6 of BEEBUG, plus the bibliography for this issue.



Boxed in the Carpark



BEEBUG Mini-Wimp

All this for £3 (cassette), £4.75 (5" & 3.5" disc) + 50p p&p.
Back issues (5.25" disc since Vol.3 No.1, 3.5" disc since Vol.5 No.1, tapes since Vol.1 No.10) available at the same prices.

SUBSCRIPTION RATES	UK ONLY			OVERSEAS		
	5" Disc	3.5" Disc	Cassette	5" Disc	3.5" Disc	Cassette
6 months (5 issues)	£25.50	£25.50	£17.00	£30.00	£30.00	£20.00
12 months (10 issues)	£50.00	£50.00	£33.00	£56.00	£56.00	£39.00

Prices are inclusive of VAT and postage as applicable. Sterling only please.

Cassette subscriptions can be commuted to a 5.25" or 3.5" disc subscription on receipt of £1.70 per issue of the subscription left to run. All subscriptions and individual orders to:

BEEBUG, Dolphin Place, Holywell Hill, St.Albans, Herts. AL1 1EX.

The Best Deals on Archimedes From BEEBUG

The Archimedes Specialists

1 0% FINANCE

For a limited period we are able to offer 0% APR finance over 9 months on the purchase of any Archimedes. You pay no interest at all. This is a brand new scheme only available from BEEBUG. The deposit and repayments are shown below.

	Deposit	9 Payments		Deposit	9 Payments
A305 Base	£83.85	£80.00	A310 Base	£90.25	£89.00
A305 Mono	£87.35	£86.00	A310 Mono	£102.75	£94.00
A305 Colour	£106.85	£103.00	A310 Colour	£113.25	£112.00
A310M Base	£96.25	£96.00	A440 Base	£267.85	£264.00
A310M Mono	£108.75	£101.00	A440 Mono	£271.35	£270.00
A310M Colour	£119.25	£119.00	A440 Colour	£290.85	£287.00

2 TRADE IN YOUR OLD BBC, MASTER OR COMPACT FOR AN ARCHIMEDES

We will be pleased to accept your old computer (in working condition) as part exchange towards the purchase of an Archimedes. (If you use the finance scheme this will replace your initial deposit on a 305/310, so you pay nothing now). Allowances are as follows:

BBC Issue 4 No DFS	£125
BBC Issue 4 DFS	£175
BBC Issue 7 No DFS	£175
BBC Issue 7 DFS (Or B+)	£225
Master 128	£250
Compact Base System	£215

Please phone for allowances on other Compact and Master systems.

3 FREE DISCS & PC EMULATOR

Join RISC USER, the Archimedes magazine and support group, and purchase your Archimedes by Cheque, Access, Visa, Official Order or 11.5% finance and we will supply you, absolutely free, 10 3.5" discs, a lockable disc storage box, printer lead and the latest version of The PC Emulator from Acorn. Altogether you save more than £142.00.

Prices Including VAT

A305 Base £803.85	Mono £861.35	Colour £1033.85
A310 Base £891.25	Mono £948.75	Colour £1121.25
A440 Base £2643.85	Mono £2701.35	Colour £2873.85

4 11.5% FINANCE OVER 12 TO 36 MONTHS

As a Licensed Credit Broker we are able to offer finance on the purchase of any equipment, including the Archimedes. You still benefit from the free PC Emulator, discs, disc box and printer lead. (Typical APR 23% on the purchase of a 310 Colour system over 36 months.
Deposit £126.25 36 payments of £37.36).

5 DISCOUNTS FOR EDUCATION

We are able to offer attractive discounts to Education Authorities, Schools, Colleges and Health Authorities. Please write with your requirements for a quotation.

TO FIND OUT MORE PHONE OR WRITE NOW. TEL: 0727 40303

We offer a complete service, including Advice, Technical Support, Showroom, Mail Order and Repairs. Our showroom in St. Albans stocks everything available for the Archimedes. Call in for a demonstration.

Please indicate your requirements below.
Subscription to Risc User (£14.50 UK) ☐ Information Pack and Catalogue ☐ 0% Finance Form for 305/310/310M/440 Base/Mono/Colour ☐ 12-36 Months Finance Form for 305/310/310M/440 Base/Mono/Colour ☐ Trade In BBC/Master/Compact ☐ Purchase 305/310/440 Base/Mono/Colour ☐
UK Courier Delivery £7.00. Overseas please ask for a quotation.

I enclose a cheque value £.....
Please debit my Access/Visa/Connect Card No
Expiry...../..... with £.....

Name .....

Address .....

.....

Signature .....

Beebug, Dolphin Place, Holywell Hill, St. Albans, Herts AL1 1EX Tel: 0727 40303
BEEBUG – The Archimedes Specialists